# Mobile Intel® Celeron® Processor (0.18µ and 0.13µ)

## Specification Update

***May 2005***

**Notice**: The mobile Intel® Celeron® processor (0.18µ and 0.13µ) may contain design defects or errors known as errata which may cause the product to deviate from published specifications. Current characterized errata are documented in this Specification Update.

Document Number: 245421-045

INFORMATION IN THIS DOCUMENT IS PROVIDED IN CONNECTION WITH INTEL® PRODUCTS. NO LICENSE, EXPRESS OR IMPLIED, BY ESTOPPEL OR OTHERWISE, TO ANY INTELLECTUAL PROPERTY RIGHTS IS GRANTED BY THIS DOCUMENT. EXCEPT AS PROVIDED IN INTEL'S TERMS AND CONDITIONS OF SALE FOR SUCH PRODUCTS, INTEL ASSUMES NO LIABILITY WHATSOEVER, AND INTEL DISCLAIMS ANY EXPRESS OR IMPLIED WARRANTY, RELATING TO SALE AND/OR USE OF INTEL PRODUCTS INCLUDING LIABILITY OR WARRANTIES RELATING TO FITNESS FOR A PARTICULAR PURPOSE, MERCHANTABILITY, OR INFRINGEMENT OF ANY PATENT, COPYRIGHT OR OTHER INTELLECTUAL PROPERTY RIGHT. Intel products are not intended for use in medical, life saving, nuclear facility, or life sustaining applications.

Intel may make changes to specifications and product descriptions at any time, without notice.

Designers must not rely on the absence or characteristics of any features or instructions marked "reserved" or "undefined." Intel reserves these for future definition and shall have no responsibility whatsoever for conflicts or incompatibilities arising from future changes to them.

The Mobile Intel® Celeron® processor (0.18µ and 0.13µ) may contain design defects or errors known as errata which may cause the product to deviate from published specifications. Current characterized errata are available on request.

Contact your local Intel sales office or your distributor to obtain the latest specifications and before placing your product order.

Intel, Pentium, Celeron, Xeon, and the Intel logo are trademarks or registered trademarks of Intel Corporation or its subsidiaries in the United States and other countries.

Intel processor numbers are not a measure of performance. Processor numbers differentiate features within each processor family, not across different processor families. See www.intel.com/products/processor_number for details.

† Hyper-Threading Technology requires a computer system with an Intel® Pentium® 4 processor supporting HT Technology and an HT Technology enabled chipset, BIOS and operating system. Performance will vary depending on the specific hardware and software you use. See http://www.intel.com/info/hyperthreading/ for more information including details on which processors support HT Technology.

*Other names and brands may be claimed as the property of others.

Copyright © 2005, Intel Corporation. All rights reserved.

# Contents

Revision History ..... 4
Preface ..... 7
Summary of Changes ..... 13
Identification Information ..... 21
Errata ..... 25
Specification Changes ..... 57
Specification Clarifications ..... 59
Documentation Changes ..... 63

# Revision History

| Revision Number | Description | Date |
|---|---|---|
| -001 | Initial release | February 2000 |
| -002 | Revised Errata M38, M43, and M47. Added Erratum M53. Added new Specification Clarification M1. | March 2000 |
| -003 | Updated the Preface with new references; Updated “Intel Celeron Processor Mobile Module Markings” section; Updated Identification Information for BGA2, micro-PGA2 packages, and mobile modules; Updated Erratum M34; Added Erratum M54; Added Documentation Change M5; Added Specification Clarifications M2, M3. | April 2000 |
| -004 | Updated Identification Information for mobile modules. Updated Erratum M53. Added Erratum M55, M56 | May 2000 |
| -005 | Updated the Preface with new document references; Updated Identification Information for BGA2, micro-PGA2 packages, and mobile modules. Added Erratum M57. | June 2000 |
| -006 | Updated Identification Information for BGA2, micro-PGA2 packages, and mobile modules; Updated Summary of Changes Tables to include C0-step products; Added Erratum M58, M59; Updated the Specification Clarifications and Documentation Changes section by removing old items that were incorporated in the new documents referenced in this spec update; Added new Specification Clarifications M1, M2. | July 2000 |
| -007 | Added Erratum M60. | August 2000 |
| -008 | Added Erratum M61, M62; Revised Erratum M22, M43, M52; Added Documentation Changes M5, M6. | September 2000 |
| -009 | Updated the list of referenced documents in the preface; Updated Identification Information for BGA2, micro-PGA2 packages, and mobile modules; Added Erratum M63; Added Documentation Changes M7, M8 | October 2000 |
| -010 | Added Erratum M64. | November 2000 |
| -011 | Updated Specification Update product key to include the Intel® Pentium® 4 processor, Revised Erratum M2; Added Documentation Changes M9 thru M14 | December 2000 |
| -012 | Revised Erratum M2; Added Documentation Changes M15, M16. | January 2001 |
| -013 | Updated the list of referenced documents in the preface; Updated Identification Information for BGA2 packages; Revised Documentation Change M15 and Added M17. | February 2001 |
| -014 | Added Erratum M65 and M66. Revised Specification Clarification M2. | March 2001 |
| -015 | Updated the list of referenced documents in the preface; Updated Identification Information for BGA2 and micro-PGA2 packages; Updated the Specification Clarifications section by removing old items that were incorporated in the new documents referenced in this spec update. | April 2001 |
| -016 | Updated the list of referenced documents in the preface; Updated Identification Information for BGA2 and micro-PGA2 packages; Changed “NoFix” plan for Erratum K38 to “Fixed” in D0 stepping; Added Erratum M67. | May 2001 |
| -017 | Updated Summary of Changes. | June 2001 |
| -018 | Updated the list of referenced documents in the preface; Updated Identification Information for BGA2 and micro-PGA2 packages. | July 2001 |

| Revision Number | Description | Date |
|---|---|---|
| -019 | Updated Summary of Changes; Added Erratum M68 and M69; Added Documentation Change M18. | August 2001 |
| -020 | Updated list of referenced documents in the preface; Updated the Celeron® mark to a registered trademark; Added micro-FCPGA and micro-FCBGA package marking diagrams for Mobile Intel Celeron Processor (0.18µ and 0.13µ) to general information section; Added identification information for Mobile Intel Celeron Processor (0.18µ and 0.13µ) micro-FCPGA and micro-FCBGA packages; Updated Summary of Changes; Updated columns with FBD0, FPD0, FPA1, FBA1 steppings in Summary of Changes; Added Errata M2AP and M70; Updated the Documentation Changes by removing old items that were incorporated in the new documents references in this spec update; Added Specification Clarification M1. | October 2001 |
| -021 | Updated Summary of Changes; Added Documentation Changes M1, M2, M3, M4, M5. | November 2001 |
| -022 | Updated the list of referenced documents in the preface. | December 2001 |
| -023 | Updated Identification Table; Added Documentation Changes M6, M7, M8, M9, and M10. | January 2002 |
| -024 | Added new Mobile Intel Celeron Processor (0.13µ) identification information | February 2002 |
| -025 | Updated Summary of Changes; Added erratum M71; updated erratum M68; added documentation change M11. | March 2002 |
| -026 | Added documentation change M1; Updated the Documentation Changes summary section by removing old items that already have been incorporated in the published Software Developer's manual. | April 2002 |
| -027 | Updated Summary of Changes; Added documentation change M2, M3, and M4; Updated Errata M48; Updated Packages Identification Information table. | May 2002 |
| -028 | Updated Summary of Changes; Added erratum M72; added Documentation changes M5 and M6; Added columns for FBB1 & FPB1. | June 2002 |
| -029 | Updated Summary of Changes; Removed old items that have been added to the Software Developers Manual; Added Documentation Changes M3, M4, M5, M6, M7, M8, M9, M10, M11, and M12. | July 2002 |
| -030 | Updated the Documentation Changes summary section by removing old items that already have been incorporated in the published Software Developer's manual | August 2002 |
| -031 | Added Documentation Changes M3, M4, M5, M6, M7, M8, M9, M10, M11, M12, M13, M14, M15, M16, M17, M18, M19, M20, M21, M22, M23, and M24. | September 2002 |
| -032 | Updated Summary of Changes; Added prefix letter W; Added Documentation Changes M25, M26, M27, M28, M29, M30, M31, and M32. | October 2002 |
| -033 | Updated the Documentation Changes summary section by removing old items that already have been incorporated in the published Software Developer's manual. Updated Identification information table. Added a note to Documentation Changes. | November 2002 |
| -034 | Updated a note in Documentation Changes section. | December 2002 |
| -035 | Updated Identification information table. | January 2003 |
| -036 | Added Erratum M73. | March 2003 |
| -037 | Updated Identification information Table 3; added prefix letter Y and Z; removed prefix letter W. | June 2003 |
| -038 | Added Errata M74 and M75. | November 2003 |
| -039 | Added errata M76. Update Errata M75. | December 2003 |

| Revision Number | Description | Date |
|---|---|---|
| -040 | Added errata M77-79 | October 2004 |
| -041 | Added errata M80-82<br>Updated errata M79 | November 2004 |
| -042 | Added errata M83-84 | December 2004 |
| -043 | Updated erratum M68 | March 2005 |
| -044 | Updated processor identification table<br>Added specification clarification M2 | April 2005 |
| -045 | Added errata M85 | May 2005 |

§

# Preface

This document is an update to the specifications contained in the documents listed in the following Affected Documents table. It is a compilation of device and document errata and specification clarifications and changes, and is intended for hardware system manufacturers and for software developers of applications, operating system, and tools.

Information types defined in the Nomenclature section of this document are consolidated into this update document and are no longer published in other documents. This document may also contain information that has not been previously published.

## Affected Documents

| Document Title | Document Number/Location |
|---|---|
| *Mobile Intel® Celeron® Processor in BGA2 and Micro-PGA2 Packages at 900 MHz, 850 MHz, 800 MHz, 750 MHz, 700 MHz, 650 MHz, 600 MHz, 550 MHz, 500 MHz, 450 MHz, Low voltage 600 MHz, Low voltage 500 MHz, Low voltage 400A MHz, Ultra Low Voltage 600MHz and Ultra Low Voltage 500 MHz datasheet* | 283654-003 |
| *Mobile Intel® Celeron® Processor (0.18µ) in Micro-FCBGA and Micro-FCPGA packages at 933, 866, 800A, and 733 MHz* | 298514-001 |
| *Mobile Intel® Celeron® Processor (0.13µ) in Micro-FCBGA in Low Voltage Package at 650 MHz* | 298517-001 |
| *Intel® Celeron® Processor Mobile Module: Mobile Module Connector 2 (MMC-2) at 700 MHz, 650 MHz, 600 MHz, 550 MHz, 500 MHz and 450 MHz datashee* | 243357-005 |
| *Intel Architecture Software Developer's Manual, Volumes 1, 2, and 3* | 243190, 243191, and 243192, respectively |
| *P6 Family of Processors Hardware Developer's Manual* | 244001 |
| *Intel® Celeron® Processor – Low Power/Ultra Low Power Datasheet* | 273509-01 |

**NOTE:** Documentation changes for IA-32 Intel® Architecture Software Developer's Manual volumes 1, 2, and 3 are posted in a separate document IA-32 Intel® Architecture Software Developer's Manual Documentation Changes. This document has been posted to http://developer.intel.com/.

# Nomenclature

**S-Spec Number** is a five-digit code used to identify products. Products are differentiated by their unique characteristics, e.g., core speed, L2 cache size, package type, etc. as described in the processor identification information table. Care should be taken to read all notes associated with each S-Spec number.

**Errata** are design defects or errors. Errata may cause the processor's behavior to deviate from published specifications. Hardware and software designed to be used with any given processor must assume that all errata documented for that processor are present on all devices unless otherwise noted.

**Documentation Changes** include errors (including typographical), or omissions from the current published specifications. These changes will be incorporated in the next release of the appropriate documentation(s).

**Specification Clarifications** describe a specification in greater detail or further highlight a specification's impact to a complex design situation. These clarifications will be incorporated in the next release of the appropriate documentation(s).

**Specification Changes** are modifications to the current published specifications for the processor. These changes will be incorporated in the next release of the appropriate documentation(s).

***Note:*** Errata remain in the specification update throughout the product's lifecycle, or until a particular stepping is no longer commercially available. Under these circumstances, errata removed from the specification update are archived and available upon request. Specification changes, specification clarifications and documentation changes are removed from the specification update when the appropriate changes are made to the appropriate product specification or user documentation (datasheets, manuals, etc.).

# General Information

**Figure 1. Mobile Intel® Celeron® Processor (Micro-PGA2) Markings**

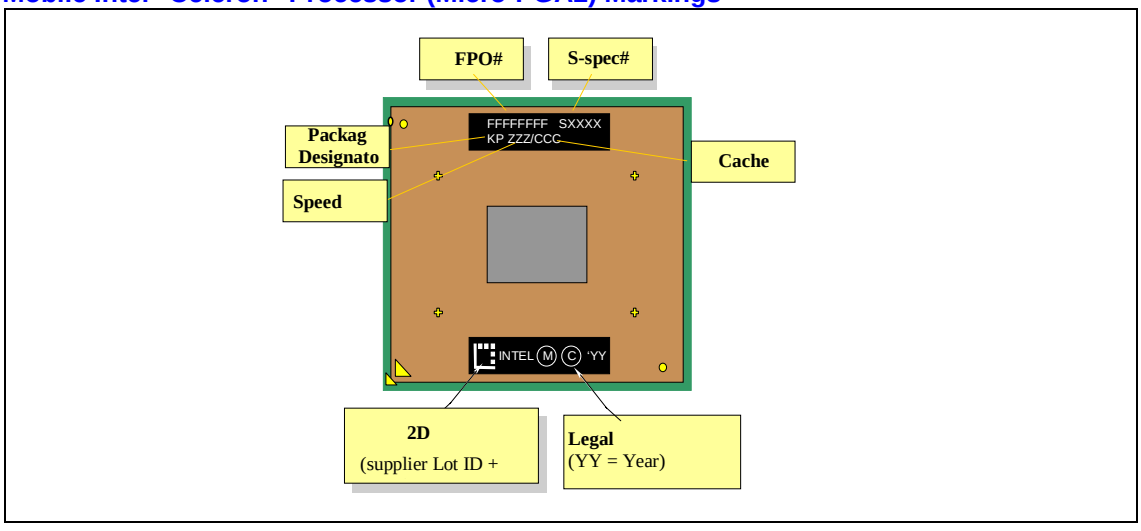


**Figure 2. Mobile Intel® Celeron® Processor (BGA2) Markings**

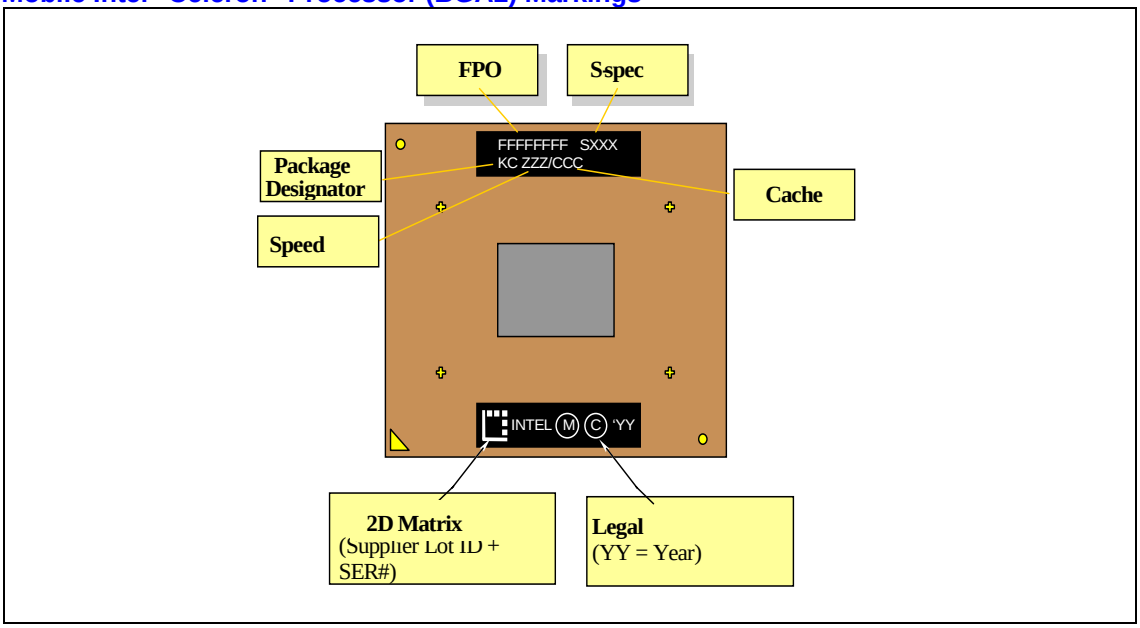



**Figure 3. Mobile Intel® Celeron® Processor 0.18µ (Micro-FCPGA) Markings**

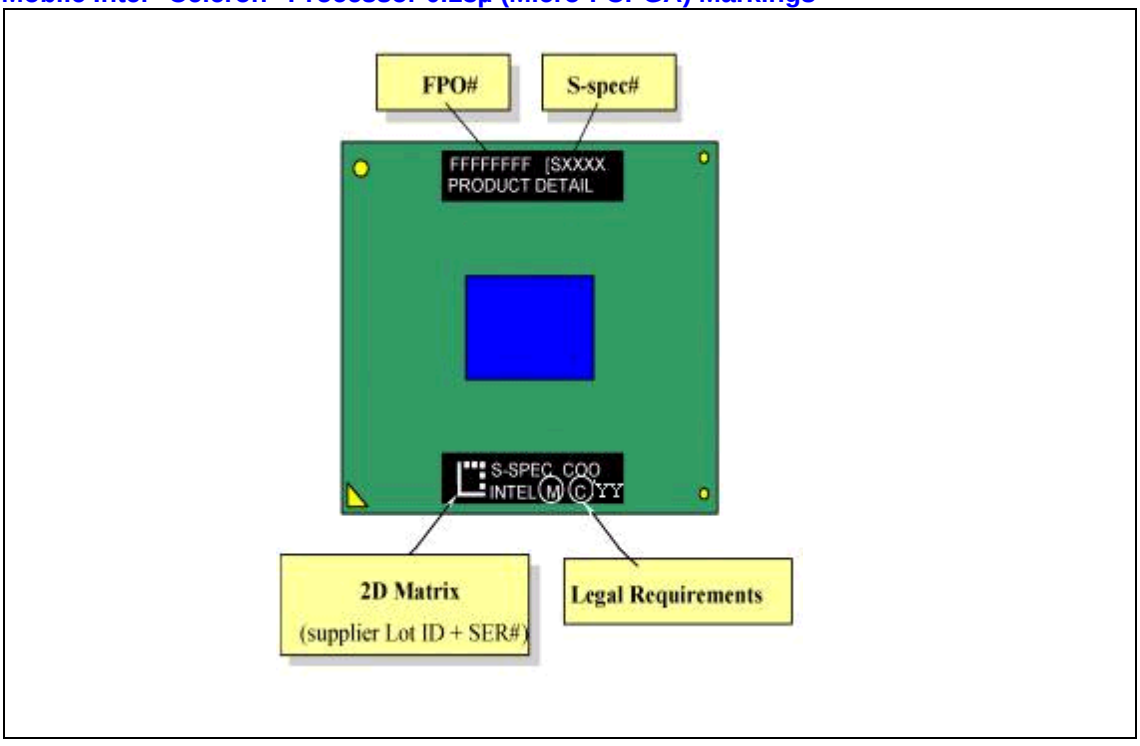


**Figure 4. Mobile Intel® Celeron® Processor 0.18µ (Micro-FCBGA) Markings**

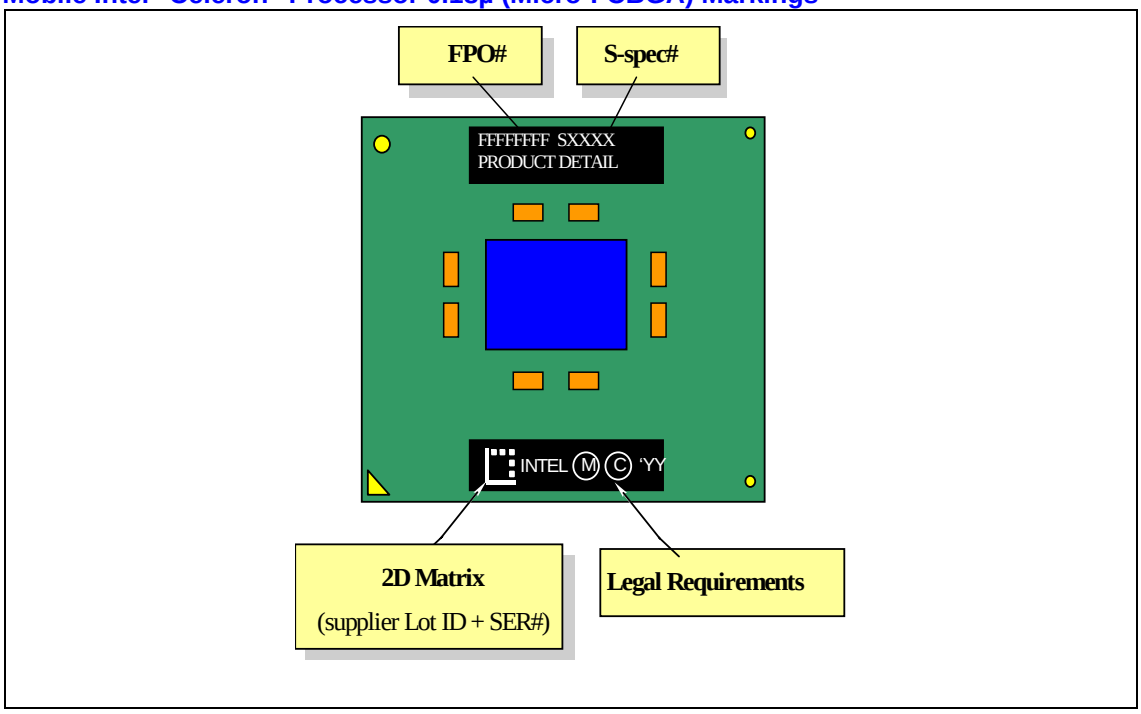



**Figure 5. Mobile Intel® Celeron® Processor 0.13µ (Micro-FCPGA) Markings**

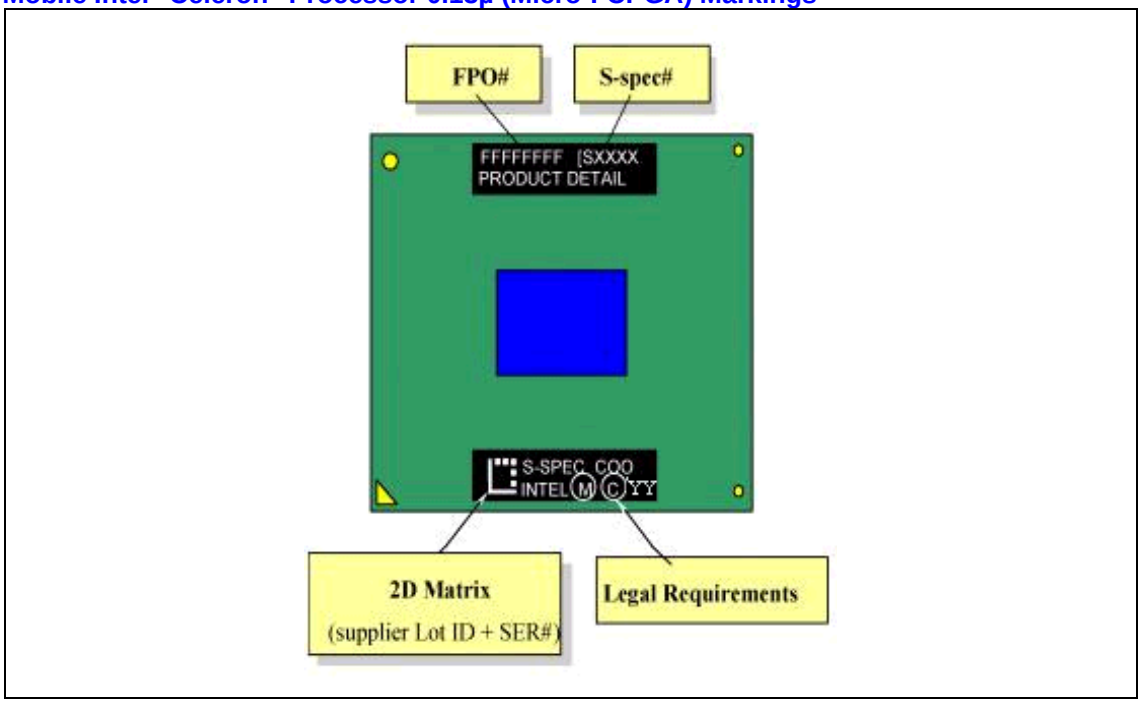


**Figure 6. Mobile Intel® Celeron® Processor 0.13µ (Micro-FCBGA) Markings**

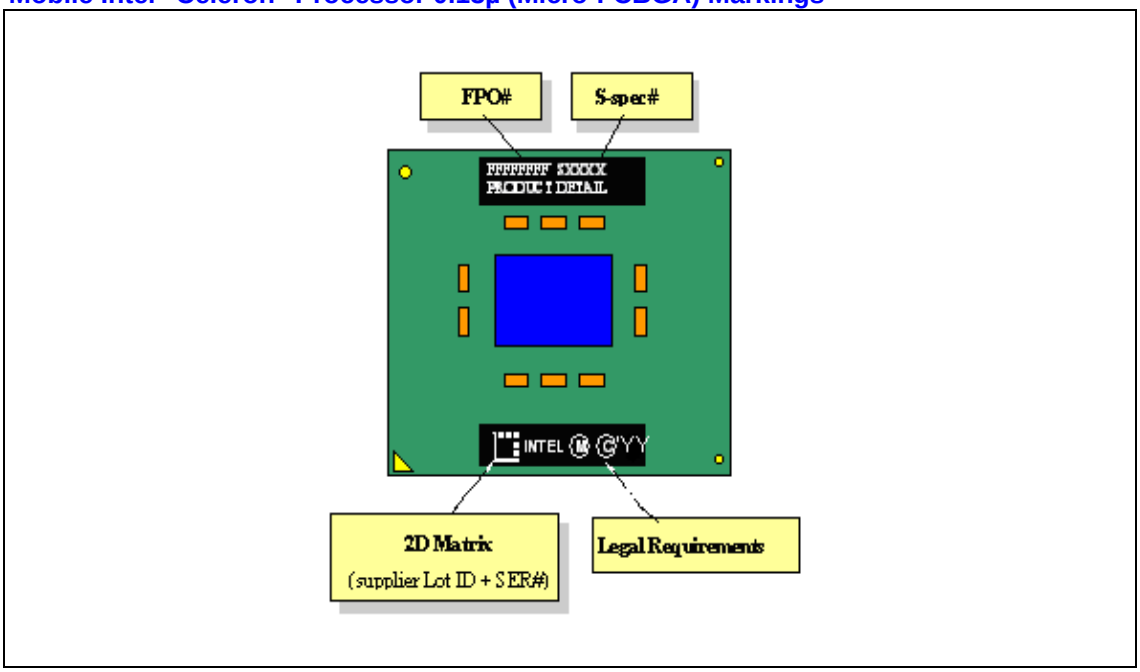
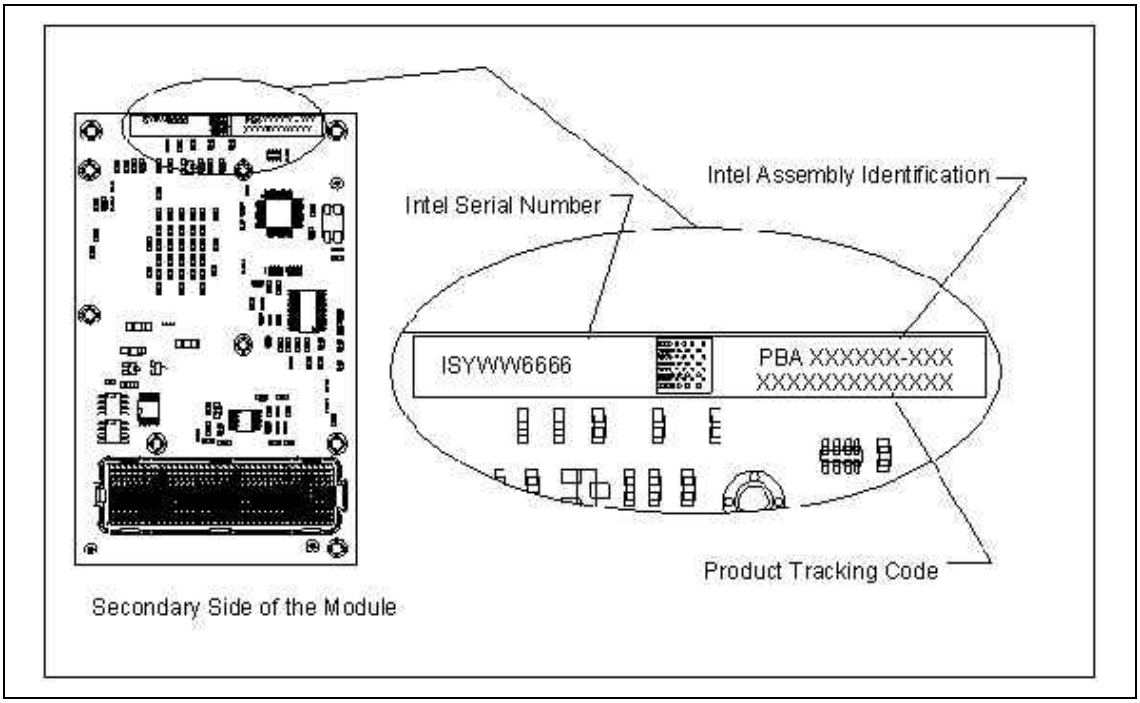



**Figure 7. Intel® Celeron® Processor Mobile Module at 650 MHz, 600 MHz, 550 MHz, 500 MHz and 450 MHz**




The Product Tracking Code (PTC) determines the Intel assembly level of the module. The PTC is on the secondary side of the module and provides the following information:

**Example:** PMN70001201AA

The PTC will consist of 13 characters as identified in the above example and can be broken down as follows:

**AABCCCDDEEEFF**

Definition:

| | | |
|---|---|---|
| AA | - | Processor Module = PM |
| B | - | Celeron® Processor (.18µ) Mobile Module (MMC-2) = N |
| CCC | - | Speed Identity = 700, 650, 600, 550, 500 or 450, etc. |
| DD | - | Cache Size = 01 (128 KB) |
| EEE | - | Notifiable Design Revision (Start at 001) |
| FF | - | Notifiable Processor Revision (Start at AA) |

For other Intel Mobile Modules, the second field (B) is defined as:
Pentium® II Processor Mobile Module (MMC-1) = D
Pentium® II Processor Mobile Module (MMC-2) = E
Pentium® II Processor Mobile Module With On-die Cache (MMC-1) = F
Pentium® II Processor Mobile Module With On-die Cache (MMC-2) = G
Celeron® Processor Mobile Module (MMC-1) = H
Celeron® Processor Mobile Module (MMC-2) = I
Pentium® III Processor Mobile Module = L
Pentium® III Processor Mobile Module Featuring Intel® SpeedStep™ Technology = M

§

# Summary of Changes

The following table indicates the Errata, Documentation Changes, Specification Clarifications, or Specification Changes that apply to the Intel Mobile Celeron processor. Intel intends to fix some of the errata in a future stepping of the component and to account for the other outstanding issues through documentation or specification changes as noted. This table uses the following notations:

## Codes Used in Summary Table

### Stepping

| | |
|---|---|
| X: | Erratum, Specification Change or Clarification that applies to this stepping. |
| (No mark) or (Blank Box): | This erratum is fixed in listed stepping or specification change does not apply to listed stepping. |

### Status

| | |
|---|---|
| Doc: | Document change or update that will be implemented. |
| PlanFix: | This erratum may be fixed in a future of the product. |
| Fixed: | This erratum has been previously fixed. |
| NoFix: | There are no plans to fix this erratum. |
| Shaded: | This item is either new or modified from the previous version of the document. |

Each Specification Update item will be prefixed with a capital letter to distinguish the product. The key below details the letters that are used in Intel's microprocessor Specification Updates:

A = Intel® Pentium® II processor

B = Mobile Intel® Pentium® II processor

C = Intel® Celeron® processor

D = Intel® Pentium® II Xeon™ processor

E = Intel® Pentium® III processor

F = Intel® Pentium® processor Extreme Edition

G = Intel® Pentium® III Xeon™ processor

H = Mobile Intel® Celeron® processor at 466 MHz, 433 MHz, 400 MHz, 366 MHz, 333 MHz, 300 MHz, and 266 MHz

K = Mobile Intel® Pentium® III Processor

L = Intel® Celeron® D processor

M = Mobile Intel® Celeron® processor

N = Intel® Pentium® 4 processor

O = Intel® Xeon™ processor MP

P = Intel® Xeon™ processor

Q = Mobile Intel® Pentium® 4 processor supporting Hyper-Threading Technology on 90-nm technology process

R = Intel® Pentium® 4 processor on 90 nm process
S = 64-bit Intel® Xeon™ Processor with 800 MHz System Bus (1 MB and 2 MB L2 cache versions)
T = Mobile Intel® Pentium® 4 processor – M
V = Mobile Intel® Celeron® processor on .13 Micron process in Micro-FCPGA Package
W = Intel® Celeron® M processor
X = Intel® Pentium® M processor on 90nm process with 2-MB L2 Cache
Y = Intel® Pentium® M processor
Z = Mobile Intel® Pentium® 4 Processor with 533 MHz System Bus

Z = Mobile Intel® Pentium® 4 Processor with 533 MHz System Bus. The Specification Updates for the Pentium® processor, Pentium® Pro processor, and other Intel products do not use this convention.

| NO. | BA2 | PA2 | MA2 | BB0 | PB0 | MB0 | BC0 | PC0 | MC0 | BD0 | PD0 | FBDO | FPDO | FBA1 | FPA1 | FBB1 | FPB1 | Plans | ERRATA |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| M1 | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | NoFix | FP data operand pointer may be incorrectly calculated after FP access which wraps 64-Kbyte boundary in 16-bit code |
| M2 | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | NoFix | Differences exist in debug exception reporting |
| M3 | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | NoFix | Code fetch matching disabled debug register may cause debug exception |
| M4 | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | NoFix | Double ECC error on read may result in BINIT# |
| M5 | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | NoFix | FP inexact-result exception flag may not be set |
| M6 | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | NoFix | BTM for SMI will contain incorrect FROM EIP |
| M7 | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | NoFix | I/O restart in SMM may fail after simultaneous MCE |
| M8 | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | NoFix | Branch traps do not function if BTMs are also enabled |
| M9 | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | NoFix | Machine check exception handler may not always execute successfully |
| M10 | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | NoFix | MCE due to L2 parity error gives L1 MCACOD.LL |
| M11 | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | NoFix | LBER may be corrupted after some events |
| M12 | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | NoFix | BTMs may be corrupted during simultaneous L1 cache line replacement |
| M13 | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | NoFix | Near CALL to ESP creates unexpected EIP address |
| M14 | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | No Fix | Memory type undefined for non-memory operations |
| M15 | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | NoFix | FP Data operand pointer may not be zero after power on or Reset |

| NO. | BA2 | PA2 | MA2 | BB0 | PB0 | MB0 | BC0 | PC0 | MC0 | BD0 | PD0 | FBDO | FPDO | FBA1 | FPA1 | FBB1 | FPB1 | Plans | ERRATA |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| M16 | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | NoFix | MOVD following zeroing instruction can cause incorrect result |
| M17 | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | NoFix | Premature execution of a load operation prior to exception handler invocation |
| M18 | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | NoFix | Read portion of RMW instruction may execute twice |
| M19 | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | NoFix | MC2_STATUS MSR has model-specific error code and machine check architecture error code reversed |
| M20 | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | NoFix | MOV with debug register causes debug exception |
| M21 | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | NoFix | Upper four PAT entries not usable with Mode B or Mode C paging |
| M22 | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | NoFix | Data breakpoint exception in a displacement relative near call may corrupt EIP |
| M23 | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | NoFix | RDMSR and WRMSR to invalid MSR may not cause GP fault |
| M24 | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | NoFix | SYSENTER/SYSEXIT instructions can implicitly load null segment selector to SS and CS registers |
| M25 | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | NoFix | PRELOAD followed by EXTEST does not load boundary scan data |
| M26 | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | NoFix | INT 1 instruction handler execution could generate a debug exception |
| M27 | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | NoFix | Misaligned Locked access to APIC space results in a hang |
| M28 | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | NoFix | Processor may assert DRDY# on a write with no data. |
| M29 | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | NoFix | GP# Fault on WRMSR to ROB_CR_BKUPTMPDR6 |
| M30 | X | X | X | X | X | X | | | | | | | | | | | | Fixed | Machine check exception may occur due to improper line eviction in the IFU |
| M31 | X | X | X | | | | | | | | | | | | | | | Fixed | Performance counters include streaming SIMD extensions L1 prefetch |
| M32 | X | X | X | | | | | | | | | | | | | | | Fixed | Processor will erroneously report a BIST failure |
| M33 | | | | | | | | | | | | | | | | | | Fix | Internal snooping mechanism causes livelock condition |
| M34 | | | | | | | | | | | | | | | | | | Fixed | Cache coherency may be lost if snoop occurs during cache line invalidation |

| NO. | BA2 | PA2 | MA2 | BB0 | PB0 | MB0 | BC0 | PC0 | MC0 | BD0 | PD0 | FBDO | FPDO | FBA1 | FPA1 | FBB1 | FPB1 | Plans | ERRATA |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| M35 | | | | | | | | | | | | | | | | | | Fix | Extra DRDY# assertion when eviction back-to-back write combining lines |
| M36 | X | X | X | | | | | | | | | | | | | | | Fixed | ECC detection and correction issue |
| M37 | X | X | X | | | | | | | | | | | | | | | Fixed | L2_LD and L2_M_LINES_OUTM performance-monitoring counters do not work |
| M38 | X | X | X | X | X | X | X | X | X | | | | | | | | | Fixed | Snoop request may cause DBSY# hang |
| M39 | X | X | X | | | | | | | | | | | | | | | Fixed | IFU/DCU deadlock may cause system hang |
| M40 | | | | | | | | | | | | | | | | | | Fix | WBINVD may lock write out buffer |
| M41 | X | X | X | | | | | | | | | | | | | | | Fixed | L2_DBUS_BUSY performance monitoring counter will not count writes |
| M42 | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | NoFix | Lower bits of SMRAM SMBASE register cannot be written with an ITP |
| M43 | X | X | X | | | | | | | | | | | | | | | Fixed | Task switch may cause wrong PTE and PDE access bit to be set |
| M44 | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | NoFix | Unsynchronized cross-modifying code operations may cause unexpected instruction execution results |
| M45 | X | X | X | X | X | X | | | | | | | | | | | | Fixed | Deadlock May Occur Due To Illegal-Instruction/Page-Miss Combination |
| M46 | X | X | X | X | X | X | | | | | | | | | | | | Fixed | MASKMOVQ Instruction Interaction with String Operation May Cause Deadlock |
| M47 | X | X | X | | | | | | | | | | | | | | | Fixed | Noise Sensitivity Issue on Processor SMI# Pin |
| M48 | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | NoFix | MOVD, CVTSI2SS, or PINSRW Following Zeroing Instruction Can Cause Incorrect Result |
| M49 | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | NoFix | FLUSH# assertion following STPCLK# may prevent CPU clocks from stopping |
| M50 | X | X | X | | | | | | | | | | | | | | | Fixed | Intermittent failure to assert ADS# during processor power-on |
| M51 | X | X | X | | | | | | | | | | | | | | | Fixed | Floating-point exception signal may be deferred |
| M52 | X | X | X | X | X | X | X | X | X | X | X | X | X | | | | | NoFix | Floating-point exception condition may be deferred |
| M53 | | | X | | | X | | | X | | | | | | | | | NoFix | Race conditions may exist on thermal sensor SMBus collision detection/arbitration circuitry |

| NO. | BA2 | PA2 | MA2 | BB0 | PB0 | MB0 | BC0 | PC0 | MC0 | BD0 | PD0 | FBDO | FPDO | FBA1 | FPA1 | FBB1 | FPB1 | Plans | ERRATA |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| M54 | X | X | X | X | X | X | | | | | | | | | | | | Fixed | Cache line reads may result in eviction of invalid data |
| M55 | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | NoFix | Snoop probe during FLUSH# could cause L2 to be left in shared state |
| M56 | X | X | X | X | X | X | | | | | | | | | | | | Fixed | Livelock may occur due to IFU line eviction |
| M57 | X | X | X | | | | | | | | | | | | | | | Fixed | Intermittent power-on failure due to uninitialized processor internal nodes |
| M58 | X | X | X | X | X | X | | | | | | | | | | | | Fixed | Selector for the LTR/LLDT register may get corrupted |
| M59 | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | NoFix | INIT does not clear global entries in the TLB |
| M60 | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | NoFix | VM bit cleared on a double fault handler |
| M61 | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | NoFix | Memory aliasing with inconsistent A and D bits may cause processor deadlock |
| M62 | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | NoFix | Use of memory aliasing with inconsistent memory type may cause system hang |
| M63 | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | NoFix | Processor may report invalid TSS fault instead of Double fault during mode C paging |
| M64 | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | NoFix | Machine check exception may occur when interleaving code between different memory types |
| M2AP | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | NoFix | Write to mask LVT (programmed as EXTINT) will not deassert outstanding interrupt |
| M65 | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | NoFix | Wrong ESP Register Values During a Fault in VM86 Mode |
| M66 | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | NoFix | APIC ICR Write May Cause Interrupt Not to be Sent When ICR Delivery Bit Pending |
| M67 | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | NoFix | Processor Incorrectly Samples NMI Interrupt after RESET# Deassertion When Processor APIC is Hardware-Disabled |
| M68 | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | Fix | The Instruction Fetch Unit (IFU) May Fetch Instructions Based Upon Stale CR3 Data After a Write to CR3 Register |
| M69 | | | | | | | | | | X | X | X | X | | | | | NoFix | Processor Might not Exit Sleep State Properly Upon De-assertion of CPUSLP# Signal |

| NO. | BA2 | PA2 | MA2 | BB0 | PB0 | MB0 | BC0 | PC0 | MC0 | BD0 | PD0 | FBDO | FPDO | FBA1 | FPA1 | FBB1 | FPB1 | Plans | ERRATA |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| M70 | | | | | | | | | | | | | | X | X | X | X | NoFix | During Boundary Scan, BCLK Not Sampled High When DPSLP# is Asserted Low |
| M71 | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | NoFix | Under some complex conditions, the instructions in the Shadow of a JMP FAR may be Unintentionally Executed and Retired |
| M72 | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | NoFix | Processor Does not Flag #GP on Non-zero Write to Certain MSRs |
| M73 | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | NoFix | IFU/BSU Deadlock May Cause System Hang |
| M74 | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | NoFix | REP MOVS Operation in Fast string Mode Continues in that Mode When Crossing into a Page with a Different Memory Type |
| M75 | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | NoFix | The FXSAVE, STOS, or MOVS Instructions May Cause a Store Ordering Violation When Data Crosses a Page with a UC Memory Type |
| M76 | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | NoFix | POPF and POPFD Instructions that Set the Trap Flag Bit May Cause Unpredictable Processor Behavior |
| M77 | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | NoFix | Code Segment Limit Violation May Occur on 4 Gbyte Limit Check |
| M78 | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | NoFix | FST Instruction with Numeric and Null Segment Exceptions May Cause General Protection Faults to be Missed and FP Linear Address (FLA) Mismatch |
| M79 | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | NoFix | Code Segment is Wrong on SMM Handler when SMBASE is not Aligned |
| M80 | X | X | X | X | X | X | X | X | X | X | X | X | X | X | x | X | x | NoFix | Page with PAT (Page Attribute Table) Set to USWC (Uncacheable Speculative Write Combine) While Associated MTRR (Memory Type Range Register) is UC (Uncacheable) May Consolidate to UC |
| M81 | X | X | X | X | X | X | X | X | X | X | X | X | X | X | x | X | x | NoFix | Under Certain Conditions LTR (Load Task Register) Instruction May Result in System Hang |
| M82 | X | X | X | X | X | X | X | X | X | X | X | X | X | X | x | X | x | NoFix | Loading from Memory Type USWC (Uncacheable Speculative Write Combine) May Get Its Data Internally Forwarded from a Previous Pending Store |

| NO. | BA2 | PA2 | MA2 | BB0 | PB0 | MB0 | BC0 | PC0 | MC0 | BD0 | PD0 | FBDO | FPDO | FBA1 | FPA1 | FBB1 | FPB1 | Plans | ERRATA |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| M83 | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | NoFix | FXSAVE after FNINIT Without an Intervening FP (Floating Point) Instruction May Save Uninitialized Values for FDP (x87 FPU Instruction Operand (Data) Pointer Offset) and FDS (x87 FPU Instruction Operand (Data) Pointer Selector) |
| M84 | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | NoFix | FSTP (Floating Point Store) Instruction Under Certain Conditions May Result In Erroneously Setting a Valid Bit on an FP (Floating Point) Stack Register |
| M85 | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | NoFix | Invalid Entries in Page-Directory-Pointer-Table Register (PDPTR) May Cause General Protection (#GP) Exception if the Reserved Bits are Set to One |

| NO. | BA2 | PA2 | MA2 | BB0 | PB0 | MB0 | BC0 | PC0 | MC0 | BD0 | PD0 | FBDO | FPDO | FBA1 | FPA1 | FBB1 | FPB1 | Plans | DOCUMENTATION CHANGES |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | | | | | | | | | | | | | | | | | There are no Documentation Changes |

| NO. | BA2 | PA2 | MA2 | BB0 | PB0 | MB0 | BC0 | PC0 | MC0 | BDO | PDO | FBDO | FPDO | FBA1 | FPA1 | FBB1 | FPB1 | Plans | SPECIFICATION CLARIFICATIONS |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| M1 | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | Doc | Temperature Specification Clarification for Measuring Currents |
| M2 | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | Doc | Specification Clarification with Respect to Time Stamp Counter |

| NO. | BA2 | PA2 | MA2 | BB0 | PB0 | MB0 | BC0 | PC0 | MC0 | BDO | PDO | FBDO | FPDO | FBA1 | FPA1 | FBB1 | FPB1 | Plans | SPECIFICATION CHANGES |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | | | | | | | | | | | | | | | | Doc | There are no Specification Changes |

§

# Identification Information

The mobile Intel® Celeron® processor (0.18µ and 0.13µ) can be identified by the following values:

| Family[1] | Model[2] | Brand ID[3] |
|---|---|---|
| 0110 | 1000 | 00000001 |

**NOTE:**

1. The Family corresponds to bits [11:8] of the EDX register after Reset, bits [11:8] of the EAX register after the CPUID instruction is executed with a 1 in the EAX register, and the generation field of the Device ID register accessible through Boundary Scan.
2. The Model corresponds to bits [7:4] of the EDX register after Reset, bits [7:4] of the EAX register after the CPUID instruction is executed with a 1 in the EAX register, and the model field of the Device ID register accessible through Boundary Scan.
3. The Brand ID is returned by the CPUID instruction in the EBX[7:0] when CPUID is executed with the value of 1 in the EAX.

### Table 1. Mobile Intel® Celeron® Processor (0.18µ) in BGA2 and micro-PGA2 Packages Identification Information

| S-Spec | Product Stepping | CPU Signature | Speed (MHz) Core/Bus | Integrated L2 Size (Kbytes) | Package | Notes |
|---|---|---|---|---|---|---|
| SL3UL | BA2 | 0681h | 400/100 | 128 | BGA2 | 1 |
| SL43W | BB0 | 0683h | 400/100 | 128 | BGA2 | 1 |
| SL45A | BB0 | 0683h | 500/100 | 128 | BGA2 | 1 |
| SL3PD | BA2 | 0681h | 450/100 | 128 | BGA2 | 2 |
| SL43T | BB0 | 0683h | 450/100 | 128 | BGA2 | 2 |
| SL3PC | BA2 | 0681h | 500/100 | 128 | BGA2 | 2 |
| SL43Q | BB0 | 0683h | 500/100 | 128 | BGA2 | 2 |
| SL3ZE | BB0 | 0683h | 550/100 | 128 | BGA2 | 2 |
| SL4AR | BB0 | 0683h | 600/100 | 128 | BGA2 | 2 |
| SL4AD | BB0 | 0683h | 650/100 | 128 | BGA2 | 2 |
| SL4J8 | BC0 | 0686h | 400/100 | 128 | BGA2 | 1 |
| SL4JC | BC0 | 0686h | 450/100 | 128 | BGA2 | 2 |
| SL4JD | BC0 | 0686h | 500/100 | 128 | BGA2 | 2 |
| SL4J9 | BC0 | 0686h | 500/100 | 128 | BGA2 | 1 |
| SL4ZR | BC0 | 0686h | 500/100 | 128 | BGA2 | 3 |
| SL4JE | BC0 | 0686h | 550/100 | 128 | BGA2 | 2 |
| SL4JF | BC0 | 0686h | 600/100 | 128 | BGA2 | 2 |
| SL4JG | BC0 | 0686h | 650/100 | 128 | BGA2 | 2 |
| SL4GU | BC0 | 0686h | 700/100 | 128 | BGA2 | 2 |
| SL56P | BC0 | 0686h | 750/100 | 128 | BGA2 | 2 |

| S-Spec | Product Stepping | CPU Signature | Speed (MHz) Core/Bus | Integrated L2 Size (Kbytes) | Package | Notes |
|---|---|---|---|---|---|---|
| SL5DR | BD0 | 068Ah | 500/100 | 128 | BGA2 | 3 |
| SL5V5 | BD0 | 068Ah | 600/100 | 128 | BGA2 | 4 |
| SL5DS | BD0 | 068Ah | 600/100 | 128 | BGA2 | 3 |
| SL582 | BD0 | 068Ah | 600/100 | 128 | BGA2 | 1 |
| SL53V | BD0 | 068Ah | 700/100 | 128 | BGA2 | 2 |
| SL53U | BD0 | 068Ah | 750/100 | 128 | BGA2 | 2 |
| SL57X | BD0 | 068Ah | 800/100 | 128 | BGA2 | 2 |
| SL57Y | BD0 | 068Ah | 850/100 | 128 | BGA2 | 2 |
| SL5LG | PD0 | 068Ah | 300/100 | 128 | BGA2 | 1,6 |
| SL544 | PD0 | 068Ah | 400/100 | 128 | BGA2 | 1,6 |
| SL543 | PD0 | 068Ah | 500/100 | 128 | BGA2 | 1,6 |
| SL584 | PD0 | 068Ah | 800/100 | 128 | BGA2 | 2,6 |
| SL3PF | PA2 | 0681h | 450/100 | 128 | Micro-PGA2 | 2 |
| SL43U | PB0 | 0683h | 450/100 | 128 | Micro-PGA2 | 2 |
| SL3PE | PA2 | 0681h | 500/100 | 128 | Micro-PGA2 | 2 |
| SL43R | PB0 | 0683h | 500/100 | 128 | Micro-PGA2 | 2 |
| SL3ZF | PB0 | 0683h | 550/100 | 128 | Micro-PGA2 | 2 |
| SL4AP | PB0 | 0683h | 600/100 | 128 | Micro-PGA2 | 2 |
| SL4AE | PB0 | 0683h | 650/100 | 128 | Micro-PGA2 | 2 |
| SL4JS | PC0 | 0686h | 450/100 | 128 | Micro-PGA2 | 2 |
| SL4JT | PC0 | 0686h | 500/100 | 128 | Micro-PGA2 | 2 |
| SL4JU | PC0 | 0686h | 550/100 | 128 | Micro-PGA2 | 2 |
| SL4JV | PC0 | 0686h | 600/100 | 128 | Micro-PGA2 | 2 |
| SL4JW | PC0 | 0686h | 650/100 | 128 | Micro-PGA2 | 2 |
| SL4GX | PC0 | 0686h | 700/100 | 128 | Micro-PGA2 | 2 |
| SL56Q | PC0 | 0686h | 750/100 | 128 | Micro-PGA2 | 2 |
| SL53D | PD0 | 068Ah | 700/100 | 128 | Micro-PGA2 | 2 |
| SL53C | PD0 | 068Ah | 750/100 | 128 | Micro-PGA2 | 2 |
| SL584 | PD0 | 068Ah | 800/100 | 128 | Micro-PGA2 | 2 |
| SL585 | PD0 | 068Ah | 850/100 | 128 | Micro-PGA2 | 2 |
| SL5PY | PD0 | 068Ah | 900/100 | 128 | Micro-PGA2 | 5 |

**NOTE:**

VID[4:0] = 01101; $V_{CC\_CORE}$ = 1.35 V
VID[4:0] = 01000; $V_{CC\_CORE}$ = 1.60 V
VID[4:0] = 10111; $V_{CC\_CORE}$ = 1.10 V
VID[4:0] = 10101; VCC_CORE = 1.15 V
VID[4:0] = 00110; VCC_CORE = 1.70 V
Support by the Embedded Intel Architecture Division

## Table 2. Identification information for Mobile Intel® Celeron® Processor (0.18µ) Micro-FCBGA and Micro-FCPGA Packages

| S-Spec | Product Stepping | CPU Signature | Speed (MHz) Core/Bus | Integrated L2 Size (Kbytes) | Package | Notes |
|---|---|---|---|---|---|---|
| SL5SU | FPD0 | 068Ah | 933/133 | 128 | Micro-FCPGA | 1 |
| SL5SR | FBD0 | 068Ah | 933/133 | 128 | Micro-FCBGA | 1 |
| SL5Q3 | FPD0 | 068Ah | 866/133 | 128 | Micro-FCPGA | 1 |
| SL5Q2 | FBD0 | 068Ah | 866/133 | 128 | Micro-FCBGA | 1 |
| SL5ST | FPD0 | 068Ah | 800A/133 | 128 | Micro-FCPGA | 1 |
| SL5SQ | FBD0 | 068Ah | 800A/133 | 128 | Micro-FCBGA | 1 |
| SL5SS | FPD0 | 068Ah | 733/133 | 128 | Micro-FCPGA | 1 |
| SL5SP | FBD0 | 068Ah | 733/133 | 128 | Micro-FCBGA | 1 |

**NOTE:**
1. VID[4:0] = 00001; VCC_CORE = 1.70V

## Table 3. Identification information for Mobile Celeron® (0.13µ) Micro-FCBGA and Micro-FCPGA Packages

| S-Spec | Product Stepping | CPU Signature | Speed (MHz) Core/Bus | Integrated L2 Size (Kbytes) | Package | Notes |
|---|---|---|---|---|---|---|
| SL6Z9 | FPB1 | 06B4h | 1266/133 | 256 | Micro-FCPGA | 3 |
| SL63Z | FPA1 | 06B1h | 1200/133 | 256 | Micro-FCBGA | 3 |
| SL6H9 | FPB1 | 06B4h | 1200/133 | 256 | Micro-FCBGA | 3 |
| SL642 | FPA1 | 06B1h | 1133/133 | 256 | Micro-FCBGA | 3 |
| SL6H8 | FPB1 | 06B4h | 1133/133 | 256 | Micro-FCBGA | 3 |
| SL643 | FPA1 | 06B1h | 1066/133 | 256 | Micro-FCPGA | 3 |
| SL6H7 | FPB1 | 06B4h | 1066/133 | 256 | Micro-FCPGA | 3 |
| SL6AB | FPB1 | 06B4h | 1000/133 | 256 | Micro-FCPGA | 5 |
| SL6B3 | FBB1 | 06B4h | 1000/133 | 256 | Micro-FCBGA | 5 |
| SL63F | FBA1 | 06B1h | 650/100 | 256 | Micro-FCBGA | 2 |
| SL5YA | FBA1 | 06B1h | 650/100 | 256 | Micro-FCBGA | 1 |
| SL6B6 | FBB1 | 06B4h | 650/100 | 256 | Micro-FCBGA | 1 |
| SL6CY | FBB1 | 06B4h | 866/133 | 256 | Micro-FCBGA | 1 |
| SL6D2 | FBB1 | 06B4h | 733/133 | 256 | Micro-FCBGA | 2 |
| SL6CZ | FBB1 | 06B4h | 700/100 | 256 | Micro-FCBGA | 2 |
| SL6D4 | FBB1 | 06B4h | 800/133 | 256 | Micro-FCBGA | 2 |
| SL6B8 | FBB1 | 06B4h | 650/100 | 256 | Micro-FCBGA | 2 |
| SL6SE | FBB1 | 06B4h | 400/100 | 256 | Micro-FCBGA | 4,S |

**NOTE:**
1. VID[4:0] = 01100; VCC_CORE = 1.15V
2. VID[4:0] = 01101; VCC_CORE = 1.10V
3. VID[4:0] = 00110; VCC_CORE = 1.45V

4. VID[4:0] = 10001; VCC_CORE = 0.95V
5. VID[4:0] = 11000; VCC_CORE = 1.40V
S. Supported by the Embedded Intel Architecture Division. Refer to Document # 273804-001

## Table 4. Intel® Celeron® Processor (0.18µ) Mobile Module Identification Information

| Product Tracking Code (PTC) | Core Stepping | CPU Signature | Speed (MHz) Core/Bus | Integrated L2 Size (Kbytes) | Package | Notes |
|---|---|---|---|---|---|---|
| PMN45001001AA | MA2 | 0681h | 450/100 | 128 | MMC2 | 1 |
| PMN50001001AA | MA2 | 0681h | 500/100 | 128 | MMC2 | 1 |
| PMN45001101AB | MB0 | 0683h | 450/100 | 128 | MMC2 | 1 |
| PMN50001101AB | MB0 | 0683h | 500/100 | 128 | MMC2 | 1 |
| PMN55001101AA | MB0 | 0683h | 550/100 | 128 | MMC2 | 1 |
| PMN60001101AA | MB0 | 0683h | 600/100 | 128 | MMC2 | 1 |
| PMN65001101AA | MB0 | 0683h | 650/100 | 128 | MMC2 | 1 |
| PMN45001201AC | MC0 | 0686h | 450/100 | 128 | MMC-2 | 1 |
| PMN50001201AC | MC0 | 0686h | 500/100 | 128 | MMC-2 | 1 |
| PMN55001201AB | MC0 | 0686h | 550/100 | 128 | MMC-2 | 1 |
| PMN60001201AB | MC0 | 0686h | 600/100 | 128 | MMC-2 | 1 |
| PMN65001201AB | MC0 | 0686h | 650/100 | 128 | MMC-2 | 1 |
| PMN70001201AA | MC0 | 0686h | 700/100 | 128 | MMC-2 | 1 |

**NOTE:** $V_{CC\_CORE}$ = 1.60 V

§

# Errata

### M1. WBINVD May Lock Write Out Buffer

**Problem:** The FP Data Operand Pointer is the effective address of the operand associated with the last noncontrol floating-point instruction executed by the machine. If an 80-bit floating-point access (load or store) occurs in a 16-bit mode other than protected mode (in which case the access will produce a segment limit violation), the memory access wraps a 64-Kbyte boundary, and the floating-point environment is subsequently saved, the value contained in the FP Data Operand Pointer may be incorrect.

**Implication:** A 32-bit operating system running 16-bit floating-point code may encounter this erratum, under the following conditions:

- The operating system is using a segment greater than 64 Kbytes in size.
- An application is running in a 16-bit mode other than protected mode.
- An 80-bit floating-point load or store which wraps the 64-Kbyte boundary is executed.
- The operating system performs a floating-point environment store (FSAVE/FNSAVE/FSTENV/FNSTENV) after the above memory access.
- The operating system uses the value contained in the FP Data Operand Pointer.

Wrapping an 80-bit floating-point load around a segment boundary in this way is not a normal programming practice. Intel has not currently identified any software which exhibits this behavior.

**Workaround:** If the FP Data Operand Pointer is used in an OS which may run 16-bit floating-point code, care must be taken to ensure that no 80-bit floating-point accesses are wrapped around a 64-Kbyte boundary.

**Status:** For the steppings affected see the *Summary of Changes* at the beginning of this section.

### M2. Differences Exist in Debug Exception Reporting

**Problem:** There exist some differences in the reporting of code and data breakpoint matches between that specified by previous Intel processor specifications and the behavior of the Intel® Mobile Celeron® processor, as described below:

**Case 1:** The first case is for a breakpoint set on a MOVSS or POPSS instruction, when the instruction following it causes a debug register protection fault (DR7.gd is already set, enabling the fault). The processor reports delayed data breakpoint matches from the MOVSS or POPSS instructions by setting the matching DR6.bi bits, along with the debug register protection fault (DR6.bd). If additional breakpoint faults are matched during the call of the debug fault handler, the processor sets the breakpoint match bits (DR6.bi) to reflect the breakpoints matched by both the MOVSS or POPSS breakpoint and the debug fault handler call. The Intel® Mobile Celeron® processor only sets DR6.bd in either situation, and does not set any of the DR6.bi bits.

**Case 2:** In the second breakpoint reporting failure case, if a MOVSS or POPSS instruction with a data breakpoint is followed by a store to memory which:

a. Crosses a 4-Kbyte page boundary,

   OR

b. Causes the page table Access or Dirty (A/D) bits to be modified,the breakpoint information for the MOVSS or POPSS will be lost. Previous processors retain this information under these boundary conditions.

**Case 3:** If they occur after a MOVSS or POPSS instruction, the INT*n*, INTO, and INT3 instructions zero the DR6.bi bits (bits B0 through B3), clearing pending breakpoint information, unlike previous processors.

**Case 4:** If a data breakpoint and an SMI (System Management Interrupt) occur simultaneously, the SMI will be serviced via a call to the SMM handler, and the pending breakpoint will be lost.

**Case 5:** When an instruction that accesses a debug register is executed, and a breakpoint is encountered on the instruction, the breakpoint is reported twice.

**Case 6:** Unlike previous versions of Intel Architecture processors, Intel® Mobile Celeron® processors will not set the Bi bits for a matching disabled breakpoint unless at least one other breakpoint is enabled.

**Implication:** When debugging or when developing debuggers for a Intel® Mobile Celeron® processor-based system, this behavior should be noted. Normal usage of the MOVSS or POPSS instructions (i.e., following them with a MOV ESP) will not exhibit the behavior of cases 1-3. Debugging in conjunction with SMM will be limited by case 4.

**Workaround:** Following MOVSS and POPSS instructions with a MOV ESP instruction when using breakpoints will avoid the first three cases of this erratum. No workaround has been identified for cases 4, 5, or 6.

**Status:** For the steppings affected see the *Summary of Changes* at the beginning of this section.

### M3. Code Fetch Matching Disabled Debug Register May Cause Debug Exception

**Problem:** The bits L0-3 and G0-3 enable breakpoints local to a task and global to all tasks, respectively. If one of these bits is set, a breakpoint is enabled, corresponding to the addresses in the debug registers DR0-DR3. If at least one of these breakpoints is enabled, any of these registers are *disabled* (i.e., L*n* and G*n* are 0), and RW*n* for the disabled register is 00 (indicating a breakpoint on instruction execution), normally an instruction fetch will not cause an instruction-breakpoint fault based on a match with the address in the disabled register(s). However, if the address in a disabled register matches the address of a code fetch which also results in a page fault, an instruction-breakpoint fault will occur.

**Implication:** The bits L0-3 and G0-3 enable breakpoints local to a task and global to all tasks, respectively. If one of these bits is set, a breakpoint is enabled, corresponding to the addresses in the debug registers DR0-DR3. If at least one of these breakpoints is enabled, any of these registers are *disabled* (i.e., L*n* and G*n* are 0), and RW*n* for the disabled register is 00 (indicating a breakpoint on instruction execution), normally an instruction fetch will not cause an instruction-breakpoint fault based on a match with the address in the disabled register(s). However, if the address in a disabled register matches the address of a code fetch which also results in a page fault, an instruction-breakpoint fault will occur.

**Workaround:** The debug handler should clear breakpoint registers before they become disabled.

**Status:** For the steppings affected see the *Summary of Changes* at the beginning of this section.

### M4. Double ECC Error on Read May Result in BINIT#

**Problem:** For this erratum to occur, the following conditions must be met:

- Machine Check Exceptions (MCEs) must be enabled.
- A dataless transaction (such as a write invalidate) must be occurring simultaneously with a transaction which returns data (a normal read).
- The read data must contain a double-bit uncorrectable ECC error.

If these conditions are met, the mobile processor will not be able to determine which transaction was erroneous, and instead of generating an MCE, it will generate a BINIT#.

**Implication:** The bus will be reinitialized in this case. However, since a double-bit uncorrectable ECC error occurred on the read, the MCE handler (which is normally reached on a double-bit uncorrectable ECC error for a read) would most likely cause the same BINIT# event.

**Workaround:** Though the ability to drive BINIT# can be disabled in the mobile processor, which would prevent the effects of this erratum, overall system behavior would not improve, since the error which would normally cause a BINIT# would instead cause the machine to shut down. No other workaround has been identified.

**Status:** For the steppings affected see the *Summary of Changes* at the beginning of this section.

### M5. FP Inexact-Result Exception Flag May Not Be Set

**Problem:** When the result of a floating-point operation is not exactly representable in the destination format (1/3 in binary form, for example), an inexact-result (precision) exception occurs. When this occurs, the PE bit (bit 5 of the FPU status word) is normally set by the processor. Under certain rare conditions, this bit may not be set when this rounding occurs. However, other actions taken by the processor (invoking the software exception handler if the exception is unmasked) are not affected. This erratum can only occur if the floating-point operation which causes the precision exception is immediately followed by one of the following instructions:

- FST m32real
- FST m64real
- FSTP m32real
- FSTP m64real
- FSTP m80real
- FIST m16int
- FIST m32int
- FISTP m16int
- FISTP m32int
- FISTP m64int

Note that even if this combination of instructions is encountered, there is also a dependency on the internal pipelining and execution state of both instructions in the processor.

**Implication:** Inexact-result exceptions are commonly masked or ignored by applications, as it happens frequently, and produces a rounded result acceptable to most applications. The PE bit of the FPU status word may not always be set upon receiving an inexact-result exception. Thus, if these exceptions are unmasked, a floating-point error exception handler may not recognize that a precision exception occurred. Note that this is a "sticky" bit, i.e., once set by an inexact-result condition, it remains set until cleared by software.

**Workaround:** This condition can be avoided by inserting two NOP instructions between the two floating-point instructions.

**Status:** For the steppings affected see the *Summary of Changes* at the beginning of this section.

### M6. BTM for SMI Will Contain Incorrect FROM EIP

**Problem:** A system management interrupt (SMI) will produce a Branch Trace Message (BTM), if BTMs are enabled. However, the FROM EIP field of the BTM (used to determine the address of the instruction which was being executed when the SMI was serviced) will not have been updated for the SMI, so the field will report the same FROM EIP as the previous BTM.

**Implication:** A BTM which is issued for an SMI will not contain the correct FROM EIP, limiting the usefulness of BTMs for debugging software in conjunction with System Management Mode (SMM).

**Workaround:** None identified

**Status:** For the steppings affected see the *Summary of Changes* at the beginning of this section.

### M7. I/O Restart in SMM May Fail After Simultaneous MCE

**Problem:** If an I/O instruction (IN, INS, REP INS, OUT, OUTS, or REP OUTS) is being executed, and if the data for this instruction becomes corrupted, the mobile processor will signal a machine check exception (MCE). If the instruction is directed at a device which is powered down, the processor may also receive an assertion of SMI#. Since MCEs have higher priority, the processor will call the MCE handler, and the SMI# assertion will remain pending. However, upon attempting to execute the first instruction of the MCE handler, the SMI# will be recognized and the processor will attempt to execute the SMM handler. If the SMM handler is completed successfully, it will attempt to restart the I/O instruction, but will not have the correct machine state, due to the call to the MCE handler.

**Implication:** A simultaneous MCE and SMI# assertion may occur for one of the I/O instructions above. The SMM handler may attempt to restart such an I/O instruction, but will have corrupted state due to the MCE handler call, leading to failure of the restart and shutdown of the processor.

**Workaround:** If a system implementation must support both SMM and MCEs, the first thing the SMM handler code (when an I/O restart is to be performed) should do is check for a pending MCE. If there is an MCE pending, the SMM handler should immediately exit via an RSM instruction and allow the machine check exception handler to execute. If there is not, the SMM handler may proceed with its normal operation.

**Status:** For the steppings affected see the *Summary of Changes* at the beginning of this section.

### M8. Branch Traps Do Not Function If BTMs Are Also Enabled

**Problem:** If branch traps or branch trace messages (BTMs) are enabled alone, both function as expected. However, if both are enabled, only the BTMs will function, and the branch traps will be ignored.

**Implication:** The branch traps and branch trace message debugging features cannot be used together.

**Workaround:** If branch trap functionality is desired, BTMs must be disabled.

**Status:** For the steppings affected see the *Summary of Changes* at the beginning of this section.

### M9. Machine Check Exception Handler May Not Always Execute Successfully

**Problem:** An MCE may not always result in the successful execution of the MCE handler. However, asynchronous MCEs usually occur upon detection of a catastrophic system condition that would also hang the processor. Leaving MCEs disabled will result in the condition which caused the asynchronous MCE instead causing the processor to enter shutdown. Therefore, leaving MCEs disabled may not improve overall system behavior.

**Implication:** No workaround, which would guarantee successful MCE handler execution under this condition, has been identified.

**Workaround:** If branch trap functionality is desired, BTMs must be disabled.

**Status:** For the steppings affected see the *Summary of Changes* at the beginning of this section.

### M10. MCE Due to L2 Parity Error Gives L1 MCACOD.LL

**Problem:** If a Cache Reply Parity (CRP) error, Cache Address Parity (CAP) error, or Cache Synchronous Error (CSER) occurs on an access to the mobile processor's L2 cache, the resulting Machine Check Architectural Error Code (MCACOD) will be logged with '01' in the LL field. This value indicates an L1 cache error; the value should be '10', indicating an L2 cache error. Note that L2 ECC errors have the correct value of '10' logged.

**Implication:** An L2 cache access error, other than an ECC error, will be improperly logged as an L1 cache error in MCACOD.LL.

**Workaround:** None identified

**Status:** For the steppings affected see the *Summary of Changes* at the beginning of this section.

### M11. LBER May Be Corrupted After Some Events

**Problem:** The last branch record (LBR) and the last branch before exception record (LBER) can be used to determine the source and destination information for previous branches or exceptions. The LBR contains the source and destination addresses for the last branch or exception, and the LBER contains similar information for the last branch taken before the last exception. This information is typically used to determine the location of a branch which leads to execution of code which causes an exception. However, after a catastrophic bus condition which results in an assertion of BINIT# and the re-initialization of the buses, the value in the LBER may be corrupted. Also, after either a CALL which results in a fault or a software interrupt, the LBER and LBR will be updated to the same value, when the LBER should not have been updated.

**Implication:** The LBER and LBR registers are used only for debugging purposes. When this erratum occurs, the LBER will not contain reliable address information. The value of LBER should be used with caution when debugging branching code; if the values in the LBR and LBER are the same, then the LBER value is incorrect. Also, the value in the LBER should not be relied upon after a BINIT# event.

**Workaround:** None identified

**Status:** For the steppings affected see the *Summary of Changes* at the beginning of this section.

### M12. BTMs May Be Corrupted During Simultaneous L1 Cache Line Replacement

**Problem:** When Branch Trace Messages (BTMs) are enabled and such a message is generated, the BTM may be corrupted when issued to the bus by the L1 cache if a new line of data is brought into the L1 data cache simultaneously. Though the new line being stored in the L1 cache is stored correctly, and no corruption occurs in the data, the information in the BTM may be incorrect due to the internal collision of the data line and the BTM.

**Implication:** Although BTMs may not be entirely reliable due to this erratum, the conditions necessary for this boundary condition to occur have only been exhibited during focused simulation testing. Intel has currently not observed this erratum in a system level validation environment.

**Workaround:** None identified

**Status:** For the steppings affected see the *Summary of Changes* at the beginning of this section.

### M13. Near CALL to ESP Creates Unexpected EIP Address

**Problem:** As documented, the CALL instruction saves procedure linking information in the procedure stack and jumps to the called procedure specified with the destination (target) operand. The target operand specifies the address of the first instruction in the called procedure. This operand can be an immediate value, a general purpose register, or a memory location. When accessing an absolute address indirectly using the stack pointer (ESP) as a base register, the base value used is the value in the ESP register before the instruction executes. However, when accessing an absolute address directly using ESP as the base register, the base value used is the value of ESP *after* the return value is pushed on the stack, not the value in the ESP register *before* the instruction executed.

**Implication:** Due to this erratum, the processor may transfer control to an unintended address. Results are unpredictable, depending on the particular application, and can range from no effect to the unexpected termination of the application due to an exception. Intel has observed this erratum only in a focused testing environment. Intel has not observed any commercially available operating system, application, or compiler that makes use of or generates this instruction.

**Workaround:** If the other seven general purpose registers are unavailable for use, and it is necessary to do a CALL via the ESP register, first push ESP onto the stack, then perform an indirect call using ESP (e.g., CALL [ESP]). The saved version of ESP should be popped off the stack after the call returns.

**Status:** For the steppings affected see the *Summary of Changes* at the beginning of this section.

### M14. Memory Type Undefined for Non-memory Operations

**Problem:** The Memory Type field for nonmemory transactions such as I/O and Special Cycles are undefined. Although the Memory Type attribute for nonmemory operations logically should (and usually does) manifest itself as UC, this feature is not designed into the implementation and is therefore inconsistent.

**Implication:** Bus agents may decode a non-UC memory type for nonmemory bus transactions.

**Workaround:** Bus agents must consider transaction type to determine the validity of the Memory Type field for a transaction.

**Status:** For the steppings affected see the *Summary of Changes* at the beginning of this section.

### M15. FP Data Operand Pointer May Not Be Zero After Power On or Reset

**Problem:** The FP Data Operand Pointer, as specified, should be reset to zero upon power on or Reset by the processor. Due to this erratum, the FP Data Operand Pointer may be nonzero after power on or Reset.

**Implication:** Software which uses the FP Data Operand Pointer and count on its value being zero after power on or Reset without first executing an FINIT/FNINIT instruction will use an incorrect value, resulting in incorrect behavior of the software.

**Workaround:** Software should follow the recommendation in Section 8.2 of the Intel Architecture Software Developer's Manual, Volume 3: System Programming Guide (Order Number 243192). This recommendation states that if the FPU will be used, software-initialization code should execute an FINIT/FNINIT instruction following a hardware reset. This will correctly clear the FP Data Operand Pointer to zero.

**Status:** For the steppings affected see the *Summary of Changes* at the beginning of this section.

### M16. MOVD Following Zeroing Instruction Can Cause Incorrect Result

**Problem:** An incorrect result may be calculated after the following circumstances occur:

1. A register has been zeroed with either a SUB reg, reg instruction or an XOR reg, reg instruction,
2. A value is moved with sign extension into the same register's lower 16 bits; or a signed integer multiply is performed to the same register's lower 16 bits,
3. This register is then copied to an MMX™ technology register using the MOVD instruction prior to any other operations on the sign-extended value.

Specifically, the sign may be incorrectly extended into bits 16-31 of the MMX technology register. Only the MMX technology register is affected by this erratum.

The erratum only occurs when the 3 following steps occur in the order shown. The erratum may occur with up to 40 intervening instructions that do not modify the sign-extended value between steps 2 and 3.

1. XOR EAX, EAX
   or SUB EAX, EAX
2. MOVSX AX, BL
   or MOVSX AX, byte ptr <memory address> or MOVSX AX, BX
   or MOVSX AX, word ptr <memory address> or IMUL BL (AX implicit, opcode F6 /5)
   or IMUL byte ptr <memory address> (AX implicit, opcode F6 /5) or IMUL AX, BX (opcode 0F AF /r)
   or IMUL AX, word ptr <memory address> (opcode 0F AF /r) or IMUL AX, BX, 16 (opcode 6B /r ib)
   or IMUL AX, word ptr <memory address>, 16 (opcode 6B /r ib) or IMUL AX, 8 (opcode 6B /r ib)
   or IMUL AX, BX, 1024 (opcode 69 /r iw)
   or IMUL AX, word ptr <memory address>, 1024 (opcode 69 /r iw) or IMUL AX, 1024 (opcode 69 /r iw)
   or CBW
3. MOVD MM0, EAX

Note that the values for immediate byte/words are merely representative (i.e., 8, 16, 1024) and that any value in the range for the size may be affected. Also, note that this erratum may occur with "EAX" replaced with any 32-bit general purpose register, and "AX" with the corresponding 16-bit version of that replacement. "BL" or "BX" can be replaced with any 8-bit or 16-bit general-purpose register. The CBW and IMUL (opcode F6 /5) instructions are specific to the EAX register only.

In the example, EAX is forced to contain 0 by the XOR or SUB instructions. Since the four types of the MOVSX or IMUL instructions and the CBW instruction modify only bits 15:8 of EAX by sign extending the lower 8 bits of EAX, bits 31:16 of EAX should always contain 0. This implies that when MOVD copies EAX to MM0, bits 31:16 of MM0 should also be 0. Under certain scenarios, bits 31:16 of MM0 are not 0, but are replicas of bit 15 (the 16th bit) of AX. This is noticeable when the value in AX after the MOVSX, IMUL or CBW instruction is negative, i.e., bit 15 of AX is a 1.

When AX is positive (bit 15 of AX is a 0), MOVD will always produce the correct answer. If AX is negative (bit 15 of AX is a 1), MOVD may produce the right answer or the wrong answer depending on the point in time when the MOVD instruction is executed in relation to the MOVSX, IMUL or CBW instruction.

**Implication:** The effect of incorrect execution will vary from unnoticeable, due to the code sequence discarding the incorrect bits, to an application failure. If the MMX technology-enabled application in which MOVD is used to manipulate pixels, it is possible for one or more pixels to exhibit the wrong color or position momentarily. It is also possible for a computational application that uses the MOVD instruction in the manner described above to produce incorrect data. Note that this data may cause an unexpected page fault or general protection fault.

**Workaround:** There are two possible workarounds for this erratum:

1. Rather than using the MOVSX-MOVD, IMUL-MOVD or CBW-MOVD pairing to handle one variable at a time, use the sign extension capabilities (PSRAW, etc.) within MMX technology for operating on multiple variables. This would result in higher performance as well.

2. Insert another operation that modifies or copies the sign-extended value between the MOVSX/IMUL/CBW instruction and the MOVD instruction as in the example below:

```
XOR EAX, EAX (or SUB EAX, EAX)
MOVSX AX, BL (or other MOVSX, other IMUL or CBW instruction)
*MOV EAX, EAX
MOVD MM0, EAX
```

MOV EAX, EAX is used here as it is fairly generic. Again, EAX can be any 32-bit register.

**Status:** For the steppings affected see the *Summary of Changes* at the beginning of this section.

### M17. Premature Execution of a Load Operation Prior to Exception Handler Invocation

**Problem:** This erratum can occur with any of the following situations:

1. If an instruction that performs a memory load causes a code segment limit violation,

2. If a waiting floating-point instruction or MMX instruction that performs a memory load has a floating-point exception pending, or

3. If an MMX instruction that performs a memory load and has either CR0.EM =1 (Emulation bit set), or a floating-point Top-of-Stack (FP TOS) not equal to 0, or a DNA exception pending.

If any of the above circumstances occur it is possible that the load portion of the instruction will have executed before the exception handler is entered.

**Implication:** In normal code execution where the target of the load operation is to write back memory there is no impact from the load being prematurely executed, nor from the restart and subsequent re-execution of that instruction by the exception handler. If the target of the load is to uncached memory that has a system side-effect, restarting the instruction may cause unexpected system behavior due to the repetition of the side-effect.

**Workaround:** Code which performs loads from memory that has side-effects can effectively workaround this behavior by using simple integer-based load instructions when accessing side-effect memory and by ensuring that all code is written such that a code segment limit violation cannot occur as a part of reading from side-effect memory.

**Status:** For the steppings affected see the *Summary of Changes* at the beginning of this section.

### M18. Read Portion of RMW Instruction May Execute Twice

**Problem:** When the mobile processor executes a read-modify-write (RMW) arithmetic instruction, with memory as the destination, it is possible for a page fault to occur during the execution of the store on the memory operand after the read operation has completed but before the write operation completes.

If the memory targeted for the instruction is UC (uncached), memory will observe the occurrence of the initial load before the page fault handler and again if the instruction is restarted.

**Implication:** This erratum has no effect if the memory targeted for the RMW instruction has no side effects. If, however, the load targets a memory region that has side effects, multiple occurrences of the initial load may lead to unpredictable system behavior.

**Workaround:** Hardware and software developers who write device drivers for custom hardware that may have a side-effect style of design should use simple loads and simple stores to transfer data to and from the device. Then, the memory location will simply be read twice with no additional implications.

**Status:** For the steppings affected see the *Summary of Changes* at the beginning of this section.

### M19. MC2_STATUS MSR Has Model-Specific Error Code and Machine Check Architecture Error Code Reversed

**Problem:** The *Intel Architecture Software Developer's Manual, Volume 3: System Programming Guide*, documents that for the MCi_STATUS MSR, bits 15:0 contain the MCA (machine-check architecture) error code field, and bits 31:16 contain the model-specific error code field. However, for the MC2_STATUS MSR, these bits have been reversed. For the MC2_STATUS MSR, bits 15:0 contain the model-specific error code field and bits 31:16 contain the MCA error code field.

**Implication:** A machine check error may be decoded incorrectly if this erratum on the MC2_STATUS MSR is not taken into account.

**Workaround:** When decoding the MC2_STATUS MSR, reverse the two error fields.

**Status:** For the steppings affected see the *Summary of Changes* at the beginning of this section.

### M20. MOV With Debug Register Causes Debug Exception

**Problem:** When in V86 mode, if a MOV instruction is executed on debug registers, a general-protection exception (#GP) should be generated, as documented in the Intel Architecture Software Developer's Manual, Volume 3: System Programming Guide, Section 14.2. However, in the case when the general detect enable flag (GD) bit is set, the observed behavior is that a debug exception (#DB) is generated instead.

**Implication:** With debug-register protection enabled (i.e., the GD bit set), when attempting to execute a MOV on debug registers in V86 mode, a debug exception will be generated instead of the expected general-protection fault.

**Workaround:** In general, operating systems do not set the GD bit when they are in V86 mode. The GD bit is generally set and used by debuggers. The debug exception handler should check that the exception did not occur in

V86 mode before continuing. If the exception did occur in V86 mode, the exception may be directed to the general-protection exception handler.

**Status:** For the steppings affected see the *Summary of Changes* at the beginning of this section.

### M21. Upper Four PAT Entries Not Usable With Mode B or Mode C Paging

**Problem:** The Page Attribute Table (PAT) contains eight entries, which must all be initialized and considered when setting up memory types for the mobile processor. However, in Mode B or Mode C paging, the upper four entries do not function correctly for 4-Kbyte pages. Specifically, bit seven of page table entries that translate addresses to 4-Kbyte pages should be used as the upper bit of a three-bit index to determine the PAT entry that specifies the memory type for the page. When Mode B (CR4.PSE = 1) and/or Mode C (CR4.PAE) are enabled, the processor forces this bit to zero when determining the memory type regardless of the value in the page table entry. The upper four entries of the PAT function correctly for 2-Mbyte and 4-Mbyte large pages (specified by bit 12 of the page directory entry for those translations).

**Implication:** Only the lower four PAT entries are useful for 4-KB translations when Mode B or C paging is used. In Mode A paging (4-Kbyte pages only), all eight entries may be used. All eight entries may be used for large pages in Mode B or C paging.

**Workaround:** None identified

**Status:** For the steppings affected see the *Summary of Changes* at the beginning of this section.

### M22. Data Breakpoint Exception in a Displacement Relative Near Call May Corrupt EIP

**Problem:** If a misaligned data breakpoint is programmed to the same cache line as the memory location where the stack push of a near call is performed and any data breakpoints are enabled, the processor will update the stack and ESP appropriately, but may skip the code at the destination of the call. Hence, program execution will continue with the next instruction immediately following the call, instead of the target of the call.

**Implication:** The failure mechanism for this erratum is that the call would not be taken; therefore, instructions in the called subroutine would not be executed. As a result, any code relying on the execution of the subroutine will behave unpredictably.

**Workaround:** Whether enabled or not, do not program a misaligned data breakpoint to the same cache line on the stack where the push for the near call is performed.

**Status:** For the stepping affected see the *Summary of Changes* at the beginning of this section.

### M23. RDMSR or WRMSR to Invalid MSR Address May Not Cause GP Fault

**Problem:** The RDMSR and WRMSR instructions allow reading or writing of MSRs (Model Specific Registers) based on the index number placed in ECX. The processor should reject access to any reserved or unimplemented MSRs by generating #GP(0). However, there are some invalid MSR addresses for which the processor will not generate #GP(0).

**Implication:** For RDMSR, undefined values will be read into EDX:EAX. For WRMSR, undefined processor behavior may result.

**Workaround:** Do not use invalid MSR addresses with RDMSR or WRMSR.

**Status:** For the steppings affected see the *Summary of Changes* at the beginning of this section.

### M24. SYSENTER/SYSEXIT Instructions Can Implicitly Load "Null Segment Selector" to SS and CS Registers

**Problem:** According to the processor specification, attempting to load a null segment selector into the CS and SS segment registers should generate a General Protection Fault (#GP). Although loading a null segment selector to the other segment registers is allowed, the processor will generate an exception when the segment register holding a null selector is used to access memory.

However, the SYSENTER instruction can implicitly load a null value to the SS segment selector. This can occur if the value in SYSENTER_CS_MSR is between FFF8h and FFFBh when the SYSENTER instruction is executed. This behavior is part of the SYSENTER/SYSEXIT instruction definition; the content of the SYSTEM_CS_MSR is always incremented by 8 before it is loaded into the SS. This operation will set the null bit in the segment selector if a null result is generated, but it does not generate a #GP on the SYSENTER instruction itself. An exception will be generated as expected when the SS register is used to access memory, however.

The SYSEXIT instruction will also exhibit this behavior for both CS and SS when executed with the value in SYSENTER_CS_MSR between FFF0h and FFF3h, or between FFE8h and FFEBh, inclusive.

**Implication:** These instructions are intended for operating system use. If this erratum occurs (and the OS does not ensure that the processor never has a null segment selector in the SS or CS segment registers), the processor's behavior may become unpredictable, possibly resulting in system failure.

**Workaround:** Do not initialize the SYSTEM_CS_MSR with the values between FFF8h and FFFBh, FFF0h and FFF3h, or FFE8h and FFEBh before executing SYSENTER or SYSEXIT.

**Status:** For the steppings affected see the *Summary of Changes* at the beginning of this section.

### M25. PRELOAD Followed by EXTEST Does Not Load Boundary Scan Data

**Problem:** According to the IEEE 1149.1 Standard, the EXTEST instruction would use data "typically loaded onto the latched parallel outputs of boundary-scan shift-register stages using the SAMPLE/PRELOAD instruction prior to the selection of the EXTEST instruction." As a result of this erratum, this method cannot be used to load the data onto the outputs.

**Implication:** Using the PRELOAD instruction prior to the EXTEST instruction will not produce expected data after the completion of EXTEST.

**Workaround:** None identified

**Status:** For the steppings affected see the *Summary of Changes* at the beginning of this section.

### M26. INT 1 Instruction Handler Execution Could Generate a Debug Exception

**Problem:** If the processor's general detect enable flag is set and an explicit call is made to the interrupt procedure via the INT 1 instruction, the general detect enable flag should be cleared prior to entering the handler. As a result of this erratum, the flag is not cleared prior to entering the handler. If an access is made to the debug registers while inside of the handler, the state of the general detect enable flag will cause a second debug exception to be taken. The second debug exception clears the general detect enable flag and returns control to the handler which is now able to access the debug registers.

**Implication:** This erratum will generate an unexpected debug exception upon accessing the debug registers while inside of the INT 1 handler.

**Workaround:** Ignore the second debug exception that is taken as a result of this erratum.

**Status:** For the steppings affected see the *Summary of Changes* at the beginning of this section.

### M27. Misaligned Locked Access to APIC Space Results in Hang

**Problem:** When the processor's APIC space is accessed with a misaligned locked access a machine check exception is expected. However, the processor's machine check architecture is unable to handle the misaligned locked access.

If this erratum occurs the processor will hang. Typical usage models for the APIC address space do not use locked accesses. This erratum will not affect systems using such a model.

**Workaround:** Ensure that all accesses to APIC space are aligned.

**Status:** For the steppings affected see the *Summary of Changes* at the beginning of this section.

### M28. Processor May Assert DRDY# on a Write with No Data

**Problem:** When a MASKMOVQ instruction is misaligned across a chunk boundary in a way that one chunk has a mask of all 0's, the processor will initiate two partial write transactions with one having all byte enables deasserted. Under these conditions, the expected behavior of the processor would be to perform both write transactions, but to deassert DRDY# during the transaction which has no byte enables asserted. As a result of this erratum, DRDY# is asserted even though no data is being transferred.

**Implication:** The implications of this erratum depend on the bus agent's ability to handle this erroneous DRDY# assertion. If a bus agent cannot handle a DRDY# assertion in this situation, or attempts to use the invalid data on the bus during this transaction, unpredictable system behavior could result.

**Workaround:** A system which can accept a DRDY# assertion during a write with no data will not be affected by this erratum. In addition, this erratum will not occur if the MASKMOVQ is aligned.

**Status:** For the steppings affected see the *Summary of Changes* at the beginning of this section.

### M29. GP# Fault on WRMSR to ROB_CR_BKUPTMPDR6

**Problem:** Writing a '1' to unimplemented bit(s) in the ROB_CR_BKUPTMPDR6 MSR (offset 1E0h) will result in a general protection fault (GP#).

**Implication:** The normal process used to write an MSR is to read the MSR using RDMSR, modify the bit(s) of interest, and then to write the MSR using WRMSR. Because of this erratum, this process may result in a GP# fault when used to modify the ROB_CR_BKUPTMPDR6 MSR.

**Workaround:** When writing to ROB_CR_BKUPTMPDR6 all unimplemented bits must be '0.' Implemented bits may be set as '0' or '1' as desired.

**Status:** For the steppings affected see the *Summary of Changes* at the beginning of this section.

### M30. Machine Check Exception May Occur Due to Improper Line Eviction in the IFU

**Problem:** The mobile processor is designed to signal an unrecoverable Machine Check Exception (MCE) as a consistency checking mechanism. Under a complex set of circumstances involving multiple speculative branches and memory accesses there exists a one cycle long window in which the processor may signal a MCE in the Instruction Fetch Unit (IFU) because instructions previously decoded have been evicted from the IFU. The one cycle long window is opened when an opportunistic fetch receives a partial hit on a previously executed but not as yet completed store resident in the store buffer. The resulting partial hit

erroneously causes the eviction of a line from the IFU at a time when the processor is expecting the line to still be present. If the MCE for this particular IFU event is disabled, execution will continue normally.

**Implication:** While this erratum may occur on a system with any number of mobile processors, the probability of occurrence increases with the number of processors. If this erratum does occur, a machine check exception will result. Note systems that implement an operating system that does not enable the Machine Check Architecture will be completely unaffected by this erratum (e.g., Windows95* and Windows98*).

**Workaround:** It is possible for BIOS code to contain a workaround for this erratum.

**Status:** For the steppings affected see the *Summary of Changes* at the beginning of this section.

### M31. Performance Counter L2 Prefetch Count Includes Streaming SIMD Extensions L1 Prefetch

**Problem:** The processors allow the measurement of the frequency and duration of numerous different internal and bus related events (see the *Intel Architecture Software Developer's Manual, Volume 3*, for more details). The Streaming SIMD Extension (SSE) architecture provides a mechanism to pre-load data into the L1 cache, bypassing the L2 cache. The number of these L1 pre-loads measured by the performance monitoring logic will incorrectly be included in the count of "L2_LINES_IN" (24H) events.

**Implication:** If application software is run which utilizes the SSE L1 prefetch feature, the count of "L2_LINES_IN" (24H) will read a value that is greater than the correct value.

**Workaround:** The correct value of this counter may be calculated by taking the value read for L2_LINES_IN (24H) and subtracting from it the value read for "EMON_KNI_PREF_MISS" (4BH, Unit Mask 00H).

**Status:** For the steppings affected see the *Summary of Changes* at the beginning of this section.

### M32. Processor Will Erroneously Report a BIST Failure

**Problem:** If the processor performs BIST at power-up, the EAX register is normally cleared (0H) if the processor passes BIST. The processor will erroneously report a non-zero value (signaling a BIST failure) even if BIST passes.

**Implication:** The processor will incorrectly signal an error after BIST is performed.

**Workaround:** The system BIOS should ignore the BIST results in the EAX register.

**Status:** For the steppings affected see the *Summary of Changes* at the beginning of this section.

### M33. Internal Snooping Mechanism Causes Livelock Condition

**Problem:** Internal timings may align where the L2 cache snooping mechanism and the Instruction Fetch Unit snooping mechanism reject each other's requests to the Data Cache Unit. Both units will continue to retry but reject requests on every other clock, leading to a livelock condition.

**Implication:** The system will hang. If an external agent is snooping the processor's caches, the hang will appear as an infinite snoop stall.

**Workaround:** It is possible for BIOS code to contain a workaround for this erratum.

**Status:** For the steppings affected see the *Summary of Changes* at the beginning of this section.

### M34. Cache Coherency May Be Lost If Snoop Occurs During Cache Line Invalidation

**Problem:** There exists a two cycle window during a cache line invalidation (due to a WBINVD instruction or FLUSH# pin assertion) during which a processor performing a snoop of that line will not see the line in the cache. In addition, when this erratum occurs, the processor invalidating the line will not write back the data in that line.

**Implication:** If this erratum occurs, cache coherency and data will be lost.

**Workaround:** None identified

**Status:** For the steppings affected see the *Summary of Changes* at the beginning of this section.

### M35. Extra DRDY# Assertion When Eviction Back-to-Back Write Combining Lines

**Problem:** The processor has the ability to evict back-to-back lines in its write combining buffers. If the processor writes back data from L1 to L2 during a back-to-back write combining line eviction, the processor may assert an extra DRDY# on the system bus.

**Implication:** Data corruption (loss of data) may occur.

**Workaround:** It is possible for BIOS code to contain a workaround for this erratum.

**Status:** For the steppings affected see the *Summary of Changes* at the beginning of this section.

### M36. Limitation on Cache Line ECC Detection and Correction

**Problem:** ECC can detect and correct up to four single-bit ECC errors per cache line. However, the processor will only detect and correct one single-bit ECC error per cache line. While all ECC errors will be detected, multiple single bit errors will be incorrectly reported as uncorrectable double bit errors, rather than correctable single bit errors.

**Implication:** The processor may report fewer single bit ECC errors and more double bit ECC errors than previous processors.

**Workaround:** None identified

**Status:** For the steppings affected see the *Summary of Changes* at the beginning of this section.

### M37. L2_LD and L2_M_LINES_OUTM Performance-Monitoring Counter Does Not Work

**Problem:** The L2_LD (29h) Performance-Monitoring counter, used for counting the number of L2 cache data loads, does not work properly.

**Implication:** This counter will report incorrect data.

**Workaround:** None identified

**Status:** For the steppings affected see the *Summary of Changes* at the beginning of this section.

### M38. Snoop Request May Cause DBSY# Hang

**Problem:** A small window of time exists in which a snoop request originating from a bus agent to a processor with one or more outstanding memory transactions may cause the processor to assert DBSY# without issuing

a corresponding bus transaction, causing the processor to hang (livelock). The exact circumstances are complex, and include the relative timing of internal processor functions with the snoop request from a bus agent.

**Implication:** This erratum may occur on a system with any number of processors. However, the probability of occurrence increases with the number of processors. If this erratum does occur, the system will hang with DBSY# asserted. At this point, the system requires a hard reset.

**Workaround:** It is possible for BIOS code to contain a workaround for this erratum.

**Status:** For the steppings affected see the *Summary of Changes* at the beginning of this section.

### M39. IFU/DCU Deadlock May Cause System Hang

**Problem:** An internal deadlock situation may occur in systems with multiple bus agents, with a failure signature such that a processor either asserts DBSY# without issuing the corresponding data, or fails to respond to a snoop request from another bus agent. Should this erratum occur, the affected processor ceases code execution and the system will hang.

The specific circumstances surrounding the occurrence of this erratum are:

1. A locked operation to the Data Cache Unit (DCU) is in process.
2. A snoop occurs, but cannot complete due to the ongoing locked operation.
3. The presence of the snoop prevents pending Instruction Fetch Unit (IFU) requests from completing.
4. The IFU requests are periodically restarted.

The continued IFU restart attempts create additional DCU snoops, which prevent the in-process locked operation from completing, keeping the DCU locked.

**Implication:** The system may hang.

**Workaround:** It is possible for BIOS code to contain a workaround for this erratum.

**Status:** For the steppings affected see the *Summary of Changes* at the beginning of this section.

### M40. WBINVD May Lock Write Out Buffer

**Problem:** If a processor is performing a WBINVD operation on a modified line, that line is stored in the processor's Write Out Buffer (WOB) until it is written to main memory. If another bus agent (such as a processor or PCI device) in the system generates a snoop that results in a hit to a modified line that is in the processor's WOB, that line could become permanently locked in the WOB. In addition to being locked in the WOB, the processor will not respond to the initial or subsequent snoop requests to this line, and the line in the WOB is never written to memory.

**Implication:** In the event of this erratum, coherency may be lost, which may result in a system lockup or system instability.

**Workaround:** None identified

**Status:** For the steppings affected see the *Summary of Changes* at the beginning of this section.

### M41. L2_DBUS_BUSY Performance Monitoring Counter Will Not Count Writes

**Problem:** The L2_DBUS_BUSY (22H) performance monitoring counter is intended to count the number of cycles during which the L2 data bus is in use. For some steppings of the processor, the L2_DBUS_BUSY counter will not be incremented during write cycles and therefore will only reflect the number of L2 data bus cycles resulting from cache reads.

**Implication:** The L2_DBUS_BUSY event counts only L2 read cycles.

**Workaround:** None identified

**Status:** For the steppings affected see the *Summary of Changes* at the beginning of this section.

### M42. Lower Bits of SMRAM SMBASE Register Cannot Be Written With an ITP

**Problem:** The System Management Base (SMBASE) register (7EF8H) stores the starting address of the System Management RAM (SMRAM). This register is used by the processor when it is in System Management Mode (SMM), and its contents serve as the memory base for code execution and data storage. The 32-bit SMBASE register can normally be programmed to any value. When programmed with an In-Target Probe (ITP), however, any attempt to set the lower 11 bits of SMBASE to anything other than zeros via the WRMSR instruction will cause the attempted write to fail.

**Implication:** When set via ITP, any attempt to relocate SMRAM space must be made with 2 KB alignment.

**Workaround:** None identified

**Status:** For the steppings affected see the *Summary of Changes* at the beginning of this section.

### M43. Task Switch May Cause Wrong PTE and PDE Access Bit to be Set

**Problem:** If an operating system executes a task switch via a Task State Segment (TSS), and the TSS is wholly or partially located within a clean page (A and D bits clear) and the GDT entry for the new TSS is either misaligned across a cache line boundary or is in a clean page, the accessed and dirty bits for an incorrect page table/directory entry may be set.

**Implication:** An operating system which uses hardware task switching (or hardware task management) may encounter this erratum. The effect of the erratum depends on the alignment of the TSS and ranges from no anomalous behavior to unexpected errors.

**Workaround:** The operating system could align all TSSs to be within page boundaries and set the A and D bits for those pages to avoid this erratum. The operating system may alternately use software task management.

**Status:** For the steppings affected see the *Summary of Changes* at the beginning of this section.

### M44. Unsynchronized Cross-Modifying Code Operations May Cause Unexpected Instruction Execution Results

**Problem:** The act of one processor, or system bus master, writing data into a currently executing code segment of a second processor with the intent of having the second processor execute that data as code is called cross-modifying code (XMC). XMC that does not force the second processor to execute a synchronizing instruction prior to execution of the new code is called unsynchronized XMC.

Software using unsynchronized XMC to modify the instruction byte stream of a processor may see unexpected instruction execution from the processor that is executing the modified code.

**Implication:** In this case, the phrase "unexpected execution behavior" encompasses the generation of most of the exceptions listed in the *Intel Architecture Software Developer's Manual Volume 3: System Programming Guide* including a General Protection Fault (GPF). In the event of a GPF the application executing the unsynchronized XMC operation would be terminated by the operating system.

**Workaround:** In order to avoid this erratum, programmers should use the XMC synchronization algorithm as detailed in the *Intel Architecture Software Developer's Manual Volume 3: System Programming Guide*, Section 7.1.3.

**Status:** For the steppings affected see the *Summary of Changes* at the beginning of this section.

### M45. Deadlock May Occur Due To Illegal-Instruction/Page-Miss Combination

**Problem:** Intel's 32-bit Instruction Set Architecture (ISA) utilizes most of the available op-code space, however some byte combinations remain undefined and are considered illegal instructions. Intel processors detect the attempted execution of illegal instructions and signal an exception. This exception is handled by operating system and/or application software.

Under a complex set of internal and external conditions involving illegal instructions, a deadlock may occur within the processor. The necessary conditions for the deadlock involve:

1. Execution of the illegal instruction.
2. Two page table walks occur within a narrow timing window coincident with the illegal instruction.

**Implication:** The illegal instructions involved in this erratum are unusual and invalid byte combinations that are not useful to application software or operating systems. These combinations are not normally generated in the course of software programming, nor are such sequences known by Intel to be generated in commercially available software and tools. Development tools (compilers, assemblers) do not generate this type of code sequence, and will normally flag such a sequence as an error. If this erratum occurs, the processor deadlock condition will occur and result in a system hang. Code execution cannot continue without a system RESET.

**Workaround:** None identified

**Status:** For the steppings affected see the *Summary of Changes* at the beginning of this section.

### M46. MASKMOVQ Instruction Interaction with String Operation May Cause Deadlock

**Problem:** Under the following scenario, combined with a specific alignment of internal events, the processor may enter a deadlock condition:

1. A store operation completes, leaving a write-combining (WC) buffer partially filled.
2. The target of a subsequent MASKMOVQ instruction is split across a cache line.
3. The data in (2) above results in a hit to the data in the WC buffer in (1).

**Implication:** If this erratum occurs, the processor deadlock condition will occur and result in a system hang. Code execution cannot continue without a system RESET.

**Workaround:** It is possible for BIOS code to contain a workaround for this erratum.

**Status:** For the steppings affected see the *Summary of Changes* at the beginning of this section.

### M47. Noise Sensitivity Issue on Processor SMI# Pin

**Problem:** Post silicon characterization has demonstrated a greater than expected sensitivity to noise on the processor's SMI# input, which may result in spurious SMI# interrupts.

**Implication:** BIOS/SMM code that is capable of handling spurious SMI events will report a spurious SMI#, but should not be negatively impacted by this erratum. Systems whose BIOS code cannot handle spurious SMI events may fail, resulting in a system hang or other anomalous behavior.

Spurious SMI# interrupts should be controlled on the system board regardless of BIOS implementation.

**Workaround:** Possible workarounds that may reduce or eliminate the occurrence of the spurious SMI include:

Use a lower effective pull-up resistance on the SMI# pin. This resistor must meet the specifications of the component driving the SMI# signal.

1. Externally condition the SMI# signal prior to providing it to the processor's SMI# pin.
2. These workarounds should be evaluated on a design-by-design basis.

**Status:** For the steppings affected see the *Summary of Changes* at the beginning of this section.

### M48. MOVD, CVTSI2SS, or PINSRW Following Zeroing Instruction Can Cause Incorrect Result

**Problem:** An incorrect result may be calculated after the following circumstances occur:

1. A register has been zeroed with either a SUB reg, reg instruction or an XOR reg, reg instruction,
2. A value is moved with sign extension into the same register’s lower 16 bits; or a signed integer multiply is performed to the same register’s lower 16 bits,
3. The register is then copied to an MMX™ technology register using the MOVD, or converted to single precision floating point and moved to an MMX technology register using the CVTSI2SS instruction prior to any other operations on the sign-extended value.

Specifically, the sign may be incorrectly extended into bits 16-31 of the MMX technology register. This erratum only affects the MMX technology register.

This erratum only occurs when the following three steps occur in the order shown below. This erratum may occur with up to 40 intervening instructions that do not modify the sign-extended value between steps 2 and 3.

1. XOR EAX, EAX
   or SUB EAX, EAX
2. MOVSX AX, BL
   or MOVSX AX, byte ptr <memory address> or MOVSX AX, BX
   or MOVSX AX, word ptr <memory address> or IMUL BL (AX implicit, opcode F6 /5)
   or IMUL byte ptr <memory address> (AX implicit, opcode F6 /5) or IMUL AX, BX (opcode 0F AF /r)
   or IMUL AX, word ptr <memory address> (opcode 0F AF /r) or IMUL AX, BX, 16 (opcode 6B /r ib)
   or IMUL AX, word ptr <memory address>, 16 (opcode 6B /r ib) or IMUL AX, 8 (opcode 6B /r ib)
   or IMUL AX, BX, 1024 (opcode 69 /r iw)
   or IMUL AX, word ptr <memory address>, 1024 (opcode 69 /r iw)
   or IMUL AX, 1024 (opcode 69 /r iw) or CBW
3. MOVD MM0, EAX or CVTSI2SS MM0, EAX

Note that the values for immediate byte/words are merely representative (i.e., 8, 16, 1024) and that any value in the range for the size is affected. Also, note that this erratum may occur with "EAX" replaced with any 32-bit general-purpose register, and "AX" with the corresponding 16-bit version of that replacement. "BL" or "BX" can be replaced with any 8-bit or 16-bit general-purpose register. The CBW and IMUL (opcode F6 /5) instructions are specific to the EAX register only.

In the above example, EAX is forced to contain 0 by the XOR or SUB instructions. Since the four types of the MOVSX or IMUL instructions and the CBW instruction only modify bits 15:8 of EAX by sign extending the lower 8 bits of EAX, bits 31:16 of EAX should always contain 0. This implies that when MOVD or CVTSI2SS copies EAX to MM0, bits 31:16 of MM0 should also be 0. In certain scenarios, bits 31:16 of MM0 are not 0, but are replicas of bit 15 (the 16th bit) of AX. This is noticeable when the value in AX after the MOVSX, IMUL or CBW instruction is negative, i.e., bit 15 of AX is a 1.

When AX is positive (bit 15 of AX is 0), MOVD or CVTSI2SS will produce the correct answer. If AX is negative (bit 15 of AX is 1), MOVD or CVTSI2SS may produce the right answer or the wrong answer, depending on the point in time when the MOVD or CVTSI2SS instruction is executed in relation to the MOVSX, IMUL or CBW instruction.

**Implication:** The effect of incorrect execution will vary from unnoticeable, due to the code sequence discarding the incorrect bits, to an application failure.

**Workaround:** There are two possible workarounds for this erratum:

1. Rather than using the MOVSX-MOVD/CVTSI2SS, IMUL-MOVD/CVTSI2SS or CBW-MOVD/CVTSI2SS pairing to handle one variable at a time, use the sign extension capabilities (PSRAW, etc.) within MMX technology for operating on multiple variables. This will also result in higher performance.

2. Insert another operation that modifies or copies the sign-extended value between the MOVSX/IMUL/CBW instruction and the MOVD or CVTSI2SS instruction as in the example below:

XOR EAX, EAX (or SUB EAX, EAX)

MOVSX AX, BL (or other MOVSX, other IMUL or CBW instruction)

*MOV EAX, EAX

MOVD MM0, EAX or CVTSI2SS MM0, EAX

*Note: MOV EAX, EAX is used here in a generic sense. Again, EAX can be substituted with any 32-bit register.

**Status:** For the steppings affected see he *Summary of Changes* at the beginning of this section.

### M49. FLUSH# Assertion Following STPCLK# May Prevent CPU Clocks From Stopping

**Problem:** If FLUSH# is asserted after STPCLK# is asserted, the cache flush operation will not occur until after STPCLK# is de-asserted. Furthermore, the pending flush will prevent the processor from entering the Sleep state, since the flush operation must complete prior to the processor entering the Sleep state.

**Implication:** Following SLP# assertion, processor power dissipation may be higher than expected. Furthermore, if the source to the processor's input bus clock (BCLK) is removed, normally resulting in a transition to the Deep Sleep state, the processor may shutdown improperly. The ensuing attempt to wake up the processor will result in unpredictable behavior and may cause the system to hang.

**Workaround:** For systems that use the FLUSH# input signal and Deep Sleep state of the processor, ensure that FLUSH# is not asserted while STPCLK# is asserted.

**Status:** For the steppings affected see the *Summary of Changes* at the beginning of this section.

### M50. Intermittent Failure to Assert ADS# during Processor Power-On

**Problem:** Under a system specific set of initial parametric conditions, a very small number of Intel® Mobile Celeron processors (CPUID 068xh) can be susceptible to entering an internal test mode during processor power-on. The symptom of this test mode is a failure to assert ADS# during a processor power-on.

**Implication:** On susceptible platforms, when power is applied to the processor, there is a possibility that the processor will occasionally enter the test mode rather than initiate a system boot sequence.

**Workaround:** A subsequent processor Power-Off then Power-On cycle should remove the processor from this test mode, allowing normal processor operation to resume.

**Status:** For the steppings affected see the *Summary of Changes* at the beginning of this section.

### M51. Floating-Point Exception Signal Can Be Deferred

**Problem:** A one clock window exists where a pending x87 FP exception that should be signaled on the execution of a CVTPS2PI, CVTPI2PS, or CVTTPS2PI instruction can be deferred to the next waiting floating-point instruction or instruction that would change MMX™ register state.

**Implication:** If this erratum occurs the floating-point exception will not be handled as expected.

**Workaround:** Applications that follow Intel programming guidelines (empty all x87 registers before executing MMX technology instructions) will not be affected by this erratum.

**Status:** For the steppings affected see the *Summary of Changes* at the beginning of this section.

### M52. Floating-Point Exception Condition May Be Deferred

**Problem:** A floating-point instruction that causes a pending floating-point exception (ES=1) is normally signaled by the processor on the next waiting FP/MMX™ technology instruction. In the following set of circumstances, the exception may be delayed or the FSW register may contain a wrong value:

1. The excepting floating-point instruction is followed by an instruction that accesses memory across a page (4-Kbyte) boundary or its access results in the update of a page table dirty/access bit.

2. The memory accessing instruction is immediately followed by a waiting floating-point or MMX technology instruction.

3. The waiting floating-point or MMX technology instruction retires during a one-cycle window that coincides with a sequence of internal events related to instruction cache line eviction.

**Implication:** The floating-point exception will not be signaled until the next waiting floating-point/MMX technology instruction. Alternatively it may be signaled with the wrong TOS and condition code values. This erratum has not been observed in any commercial software applications.

**Workaround:** None identified

**Status:** For the stepping affected see the *Summary of Changes* at the beginning of this section.

### M53. Race Conditions May Exist on Thermal Sensor SMBus Collision Detection/Arbitration Circuitry

**Problem:** In certain SMBus configurations, when the thermal sensor is used in "hard wired alert" mode along with at least one other device on the bus, the thermal sensor may continue to send its address after losing a collision arbitration in response to an Alert Response Address (ARA) by the SMBus controller.

In order for this erratum to occur, all of the following conditions must be present:

1. The thermal sensor must be configured with alert enabled (default setting).

2. There must be one or more other devices on the SMBus along with the thermal sensor.

3. One or more of these other devices must be also configured with the alert enabled.

4. One or more of these other devices must have a lower address (higher priority) than the thermal sensor.

5. The thermal sensor must generate an SM alert while at least one other device has an SM alert pending to be serviced.

In this situation, the thermal sensor will continue to send its address on the SMBus even if it has a lower priority than the pending alert. When this occurs, the SMBus controller cannot correctly interpret the device address. This may cause the thermal sensor's alert flag not to clear and may result in SMBus lockup.

**Implication:** The SMBus controller may see an invalid address and the resulting response of the SMBus controller will vary from implementation to implementation.

**Workaround:** Remove any one of the five conditions listed above or:

1. In software, use polling mode for the thermal sensor data collection with alert disabled. This software workaround has been validated on both Intel's test platforms as well as on certain OEM systems.

2. Ensure that the thermal sensor alert may be cleared by a hardware or software mechanism. The implementation of this workaround will be system dependent.

**Status:** For the steppings affected, see the *Summary of Changes* at the beginning of this section.

### M54. Cache Line Reads May Result in Eviction of Invalid Data

**Problem:** A small window of time exists in which internal timing conditions in the processor cache logic may result in the eviction of an L2 cache line marked in the invalid state.

**Implication:** There are three possible implications of this erratum:

1. The processor may provide incorrect L2 cache line data by evicting an invalid line.
2. A BNR# (Block Next Request) stall may occur on the system bus.
3. Should a snoop request occur to the same cache line in a small window of time, the processor may incorrectly assert HITM#. It is then possible for an infinite snoop stall to occur should another processor respond (correctly) to the snoop request with HIT#. In order for this infinite snoop stall to occur, at least three agents must be present, and the probability of occurrence increases with the number of processors.

Should 2 or 3 occur, the processor will eventually assert BINIT# (if enabled) with an MCA error code indicating a ROB time-out. At this point, the system requires a hard reset.

**Workaround:** It is possible for BIOS code to contain a workaround for this erratum.

**Status:** For the steppings affected see the *Summary of Changes* at the beginning of this section.

### M55. Snoop Probe During FLUSH# Could Cause L2 to be Left in Shared State

**Problem:** During a L2 FLUSH operation using the FLUSH# pin, it is possible that a read request from a bus agent or other processor to a valid line will leave the line in the Shared state (S) instead of the Invalid state (I) as expected after flush operation. Before the FLUSH operation is completed, another snoop request to invalidate the line from another agent or processor could be ignored, again leaving the line in the Shared state.

**Implication:** Current desktop and mid range server systems have no mechanism to assert the flush pin and hence are not affected by this errata. A high end server system that does not suppress snoop traffic before the assertion of the FLUSH# pin may cause a line to be left in an incorrect cache state.

**Workaround:** Affected systems (those capable of asserting the FLUSH# pin) should prevent snoop activity on the front side bus until invalidation is completed after asserting FLUSH#, or use a WBINVD instruction instead of asserting the FLUSH# pin in order to flush the cache.

**Status:** For the steppings affected see the *Summary of Changes* at the beginning of this section.

### M56. Livelock May Occur Due to IFU Line Eviction

**Problem:** Following the conditions outlined for erratum M30, if the instruction that is currently being executed from the evicted line must be restarted by the IFU, and the IFU receives another partial hit on a previously executed (but not as yet completed) store that is resident in the store buffer, then a livelock may occur.

**Implication:** If this erratum occurs, the processor will hang in a live lock-situation, and the system will require a reset to continue normal operation.

**Workaround:** None identified

**Status:** For the steppings affected see the *Summary of Changes* at the beginning of this section.

### M57. Intermittent Power-on Failure due to Uninitialized Processor Internal Nodes

**Problem:** If there is no clock source supplied to the processor's PICCLK pin, the processor may drive an incorrect address for the reset vector at power-on due to uninitialized processor internal nodes. In this scenario when ADS# is asserted, it is possible that the processor drives either the SMI or NMI vector addresses, rather than the reset vector address.

**Implication:** Systems that provide a clock to the processor's PICCLK pin are unaffected by this issue. On a system implementation with no clock source supplied to the processor's PICCLK pin, a small percentage of the systems may intermittently fail to boot, or may fail to resume from a STR or STD state. On the next power-on, the system will likely boot normally.

**Workaround:** Supply a clock source to the processor's PICCLK pin.

**Status:** For the steppings affected see the *Summary of Changes* at the beginning of this section.

### M58. Selector for the LTR/LLDT Register May Get Corrupted

**Problem:** The internal selector portion of the respective register (TR, LDTR) may get corrupted if, during a small window of LTR or LLDT system instruction execution, the following sequence of events occur:

1. Speculative write to a segment register that might follow the LTR or LLDT instruction
2. The read segment descriptor of LTR/LLDT operation spans a page (4 Kbytes) boundary; or causes a page fault

**Implication:** Incorrect selector for LTR, LLDT instruction could be used after a task switch.

**Workaround:** Software can insert a serializing instruction between the LTR or LLDT instruction and the segment register write.

**Status:** For the steppings affected see the *Summary of Changes* at the beginning of this section.

### M59. INIT Does Not Clear Global Entries in the TLB

**Problem:** INIT may not flush a TLB entry when:

1. The processor is in protected mode with paging enabled and the page global enable flag is set (PGE bit of CR4 register)
2. G bit for the page table entry is set
3. TLB entry is present in TLB when INIT occurs

**Implication:** Software may encounter unexpected page fault or incorrect address translation due to a TLB entry erroneously left in TLB after INIT.

**Workaround:** Write to CR3, CR4 or CR0 registers before writing to memory early in BIOS code to clear all the global entries from TLB.

**Status:** For the steppings affected see the *Summary of Changes* at the beginning of this section.

### M60. VM Bit Will be Cleared on a Double Fault Handler

**Problem:** Following a task switch to a Double Fault Handler that was initiated while the processor was in virtual-8086 (VM86) mode, the VM bit will be incorrectly cleared in EFLAGS.

**Implication:** When the OS recovers from the double fault handler, the processor will no longer be in VM86 mode.

**Workaround:** None identified

**Status:** For the steppings affected see the *Summary of Changes* at the beginning of this section.

### M61. Memory Aliasing with Inconsistent A and D Bits May Cause Processor Deadlock

**Problem:** In the event that software implements memory aliasing by having two Page Directory Entries(PDEs) point to a common Page Table Entry(PTE) and the Accessed and Dirty bits for the two PDEs are allowed to become inconsistent the processor may become deadlocked.

**Implication:** This erratum has not been observed with commercially available software.

**Workaround:** Software that needs to implement memory aliasing in this way should manage the consistency of the Accessed and Dirty bits

**Status:** For the steppings affected see the *Summary of Changes* at the beginning of this section.

### M62. Use of Memory Aliasing with Inconsistent Memory Type May Cause System Hang

**Problem:** Software that implements memory aliasing by having more than one linear address mapped to the same physical page with different cache types may cause the system to hang. This would occur if one of the addresses were non-cacheable used in code segment and the other a cacheable address. If the cacheable address finds its way in instruction cache, and non-cacheable address is fetched in IFU, the processor may invalidate the non-cacheable address from the fetch unit. Any micro-architectural event that causes instruction restart will expect this instruction to still be in fetch unit and lack of it will cause system hang.

**Implication:** This erratum has not been observed with commercially available software.

**Workaround:** Although it is possible to have a single physical page mapped by two different linear addresses with different memory types, Intel has strongly discouraged this practice as it may lead to undefined results. Software that needs to implement memory aliasing should manage the memory type consistency.

**Status:** For the steppings affected see the *Summary of Changes* at the beginning of this section.

### M63. Processor may Report Invalid TSS Fault Instead of Double Fault During Mode C Paging

**Problem:** When an operating system executes a task switch via a Task State Segment (TSS) the CR3 register is always updated from the new task TSS. In the mode C paging, once the CR3 is changed the processor will attempt to load the PDPTRs. If the CR3 from the target task TSS or task switch handler TSS is not valid then the new PDPTR will not be loaded. This will lead to the reporting of invalid TSS fault instead of the expected Double fault.

**Implication:** Operating systems that access an invalid TSS may get invalid TSS fault instead of a Double fault.

**Workaround:** Software needs to ensure any accessed TSS is valid.

**Status:** For the steppings affected see the *Summary of Changes* at the beginning of this section.

### M64. Machine Check Exception may Occur When Interleaving Code Between Different Memory Types

**Problem:** A small window of opportunity exists where code fetches interleaved between different memory types may cause a machine check exception. A complex set of micro-architectural boundary conditions is required to expose this window.

**Implication:** Interleaved instruction fetches between different memory types may result in a machine check exception. The system may hang if machine check exceptions are disabled. Intel has not observed the occurrence of this erratum while running commercially available applications or operating systems.

**Workaround:** Software can avoid this erratum by placing a serializing instruction between code fetches, which span different memory types.

**Status:** For the steppings affected see the *Summary of Changes* at the beginning of this section.

### M2AP. Write to Mask LVT (Programmed as EXTINT) Will Not Deassert Outstanding Interrupt

**Problem:** If the APIC subsystem is configured in Virtual Wire Mode implemented through the local APIC (i.e., the 8259 INTR signal is connected to LINT0 and LVT1's interrupt delivery mode field is programmed as EXTINT), a write to LVT1 intended to mask interrupts will not deassert the internal interrupt source if the external LINT0 signal is already asserted. The interrupt will be erroneously posted to the mobile Pentium III processor despite the attempt to mask it via the LVT.

**Implication:** Because of the masking attempt, interrupts may be generated when the system software expects no interrupts to be posted.

**Workaround:** Software can issue a write to the 8259A interrupt mask register to deassert the LINT0 interrupt level, followed by a read to the controller to ensure that the LINT0 signal has been deasserted. Once this is ensured, software may then issue the write to mask LVT entry 1.

**Status:** For the steppings affected see the *Summary of Changes* at the beginning of this section.

### M65. Wrong ESP Register Values During a Fault in VM86 Mode

**Problem:** At the beginning of the IRET instruction execution in VM86 mode, the lower 16 bits of the ESP register are saved as the old stack value. When a fault occurs, these 16 bits are moved into the 32-bit ESP, effectively clearing the upper 16 bits of the ESP.

**Implication:** This erratum has not been observed to cause any problems with commercially available software.

**Workaround:** None identified

**Status:** For the steppings affected see the *Summary of Changes* at the beginning of this section.

### M66. APIC ICR Write May Cause Interrupt Not to be Sent When ICR Delivery Bit Pending

**Problem:** If the APIC ICR (Interrupt Control Register) is written with a new interrupt command while the Delivery Status bit from a previous interrupt command is set to '1' (Send Pending), the interrupt message may not be sent out by the processor.

**Implication:** This erratum will cause an interrupt message not to be sent, potentially resulting in system hang.

**Workaround:** Software should always poll the Delivery Status bit in the APIC ICR and ensure that it is '0' (Idle) before writing a new value to the ICR.

**Status:** For the steppings affected see the *Summary of Changes* at the beginning of this section.

### M67. Processor Incorrectly Samples NMI Interrupt after RESET# Deassertion When Processor APIC is Hardware-Disabled

**Problem:** When the processor APIC is hardware-disabled the processor may incorrectly interpret the NMI signal as an NMI interrupt, instead of a frequency strap value, starting six bus clocks after RESET# is de-asserted. This will result in a processor hang due to the NMI Handler not being installed at this time.

**Implication:** The system may fail to boot due to this issue.

**Workaround:** The processor APIC must be hardware-enabled by pulling PICD[1:0] high with separate pull up resistors and supplying PICCLK to the processor.

**Status:** For the steppings affected, see the *Summary of Changes* at the beginning of this section.

### M68. The Instruction Fetch Unit (IFU) May Fetch Instructions Based Upon Stale CR3 Data After a Write to CR3 Register

**Problem:** Under a complex set of conditions, there exists a one clock window following a write to the CR3 register where-in it is possible for the iTLB fill buffer to obtain a stale page translation based on the stale CR3 data. This stale translation will persist until the next write to the CR3 register, the next page fault or execution of a certain class of instructions including CPUID or IRETD with privilege level change.

**Implication:** The wrong page translation could be used leading to erroneous software behavior.

**Workaround:** Operating systems that are potentially affected can add a second write to the CR3 register.

**Status:** For the steppings affected, see the *Summary of Changes* at the beginning of this section.

### M69. Processor Might not Exit Sleep State Properly Upon De-assertion of CPUSLP# Signal

**Problem:** If the processor enters a sleep state upon assertion of CPUSLP# signal, and if the core to system bus multiplier is an odd bus fraction, then the processor may not resume from the CPU sleep state upon the de-assertion of CPUSLP# signal.

**Implication:** This erratum may result in a system hang during a resume from CPU sleep state. Mobile platforms using Quick Start recommendations are not affected.

**Workaround:** It is possible to workaround this in BIOS by not asserting CPUSLP# for power management purposes. For mobile platforms, the workaround is to use the Quick Start recommendation.

**Status:** For the steppings affected, see the *Summary of Changes* at the beginning of this section.

### M70. During Boundary Scan, BCLK Not Sampled High When DPSLP# is Asserted Low

**Problem:** During boundary scan, BCLK not sampled high when DPSLP# is asserted low.

**Implication:** Boundary scan results may be incorrect when DPSLP# is asserted low.

**Workaround:** Do not use boundary scan when DPSLP# is asserted low.

**Status:** For the steppings affected, see the *Summary of Changes* at the beginning of this section.

### M71. Under Some Complex Conditions, the Instructions in the Shadow of a JMP FAR may be Unintentionally Executed and Retired

**Problem:** If all of the following events happen in sequence it is possible for the system or application to hang or to execute with incorrect data.

1. The execution of an instruction, with an OPCODE that requires the processor to stall the issue of micro-instructions in the flow from the microcode sequence logic block to the instruction decode block (a StallMS condition).
2. Less than 63 (39 for Pre-CPUID 0x6BX) micro-instructions later, the execution of a mispredictable branch instruction (Jcc, LOOPcc, RET Near, CALL Near Indirect, JMP ECX=0, or JMP Near Indirect).
3. The conditional branch in event (2) is mispredicted, and furthermore the mispredicted path of execution must result in either an ITLB miss, or an Instruction Cache miss. This needs to briefly stall the issue of micro-instructions again immediately after the conditional branch until that branch prediction is corrected by the jump execution block (a 2nd StallMS condition).
4. Along the correct path of execution, the next instruction must contain a 3rd StallMS condition at a precisely aligned point in the execution of the instruction ( CLTS, POPSS, LSS, or MOV to SS ).
5. A JMP FAR instruction must execute within the next 63 micro-instructions (39 Pre-CPUID 0x6BX). The intervening micro-instructions must not have any events or faults. When the instruction from event (2) retires, the StallMS condition within the event (5) instruction fails to operate correctly, and instructions in the shadow of the JMP FAR instruction could be unintentionally executed.

**Implication:** Occurrence of this erratum could lead to erroneous software behavior. Intel has not identified any commercially available software which may encounter this condition; this erratum was discovered in a focused test environment. One of the four instructions that are required to trigger this erratum, CLTS, is a privileged instruction that is only executed by an operating system or driver code. The remaining three instructions, POPSS, LSS, and MOV to SS, are executed infrequently in modern 32-bit application code.

**Workaround:** None identified at this time.

**Status:** For the stepping affected see the Summary of Changes at the beginning of this section.

### M72. Processor Does Not Flag #GP on Non-zero Write to Certain MSRs

**Problem:** When a non-zero write occurs to the upper 32 bits of SYSENTER_EIP_MSR or SYSENTER_ESP_MSR, the processor should indicate a general protection fault by flagging #GP. Due to this erratum, the processor does not flag #GP.

**Implication:** The processor unexpectedly does not flag #GP on a non-zero write to the upper 32 bits of SYSENTER_EIP_MSR or SYSENTER_ESP_MSR. No known commercially available operating system has been identified to be affected by this erratum.

**Workaround:** None identified.

**Status:** For the steppings affected see the Summary of Changes at the beginning of this section.

### M73. IFU/BSU Deadlock May Cause System Hang

**Problem:** A lockable instruction with memory operand that spans across two pages may, given some rare internal conditions, hang the system.

**Implication:** When this erratum occurs, the system may hang. Intel has not observed this erratum with any commercially available software or system.

**Workaround:** Lockable data should always be contained in a single page.

**Status:** For the steppings affected see *the Summary of Changes* at the beginning of this section.

### M74. REP MOVS Operation in Fast string Mode Continues in that Mode When Crossing into a Page with a Different Memory Type

**Problem:** A fast "REP MOVS" operation will continue to be handled in fast mode when the string operation crosses a page boundary into an Uncacheable (UC) memory type. Also if the fast string operation crosses a page boundary into a WC memory region, the processor will not self snoop the WC memory region. This may eventually result in incorrect data for the WC portion of the operation if those cache lines were previously cached as WB (through aliasing) and modified.

**Implication:** String elements should be handled by the processor at the native operand size in UC memory. In the event that the WB to WC aliasing case occurs, the end result could vary from normal software execution to potential software failure. Intel has not observed either aspects of this erratum in commercially available software.

**Workaround:** Software operating within Intel's recommendation will not require WB and WC memory aliased to the same physical address.

**Status:** For the steppings affected, see the Summary Tables of Changes.

### M75. The FXSAVE, STOS, or MOVS Instructions May Cause a Store Ordering Violation When Data Crosses a Page with a UC Memory Type

**Problem:** If the data from an FXSAVE, STOS, or MOVS instruction crosses a page boundary from WB to UC memory type and this instruction is immediately followed by a second instruction that also issues a store to memory, the final data stores from both instructions may occur in the wrong order.

**Implication:** The impact of this store ordering behavior may vary from normal software execution to potential software failure. Intel has not observed this erratum in commercially available software.

**Workaround:** FXSAVE, STOS, or MOVS data must not cross page boundary from WB to UC memory type.

**Status:** For the steppings affected, see the Summary Tables of Changes.

### M76. POPF and POPFD Instructions that Set the Trap Flag Bit May Cause Unpredictable Processor Behavior

**Problem:** In some rare cases, POPF and POPFD instructions that set the Trap Flag (TF) bit in the EFLAGS register (causing the processor to enter Single-Step mode) may cause unpredictable processor behavior.

**Implication:** Single step operation is typically enabled during software debug activities, not during normal system operation.

**Workaround:** There is no workaround for single step operation in commercially available software. For debug activities on custom software, the POPF and POPFD instructions could be immediately followed by a NOP instruction to facilitate correct execution.

**Status:** For the steppings affected, see the Summary Tables of Changes.

### M77. Code Segment Limit Violation May Occur on 4 Gigabyte Limit Check

**Problem:** Code Segment limit violation may occur on 4 Gigabyte limit check when the code stream wraps around in a way that one instruction ends at the last byte of the segment and the next instruction begins at 0x0.

**Implication:** This is a rare condition that may result in a system hang. Intel has not observed this erratum with any commercially available software, or system.

**Workaround:** Avoid code that wraps around segment limit.

**Status:** For the steppings affected, see the Summary Tables of Changes.

### M78. FST Instruction with Numeric and Null Segment Exceptions May Cause General Protection Faults to be Missed and FP Linear Address (FLA) Mismatch

**Problem:** FST instruction combined with numeric and null segment exceptions may cause General Protection Faults to be missed and FP Linear Address (FLA) mismatch.

**Implication:** This is a rare condition that may result in a system hang. Intel has not observed this erratum with any commercially available software, or system.

**Workaround:** None identified.

**Status:** For the steppings affected, see the Summary Tables of Changes.

### M79. Code Segment (CS) Is Wrong on SMM Handler when SMBASE Is Not Aligned

**Problem:** With SMBASE being relocated to a non-aligned address, during SMM entry the CS can be improperly updated which can lead to an incorrect SMM handler.

**Implication:** This is a rare condition that may result in a system hang. Intel has not observed this erratum with any commercially available software, or system.

**Workaround:** Align SMBASE to 32K byte.

**Status:** For the steppings affected, see the Summary Tables of Changes.

### M80. Page with PAT (Page Attribute Table) Set to USWC (Uncacheable Speculative Write Combine) While Associated MTRR (Memory Type Range Register) is UC (Uncacheable) May Consolidate to UC

**Problem:** For a page whose PAT memory type is USWC while the relevant MTRR memory type is UC, the consolidated memory type may be treated as UC (rather than WC as specified in IA-32 Intel® Architecture Software Developer's Manual)..

**Implication:** When this erratum occurs, the memory page may be treated as UC (rather than WC). This may have a negative performance impact.

**Workaround:** None identified.

**Status:** For the steppings affected, see the Summary Tables of Changes.

### M81. Under Certain Conditions LTR (Load Task Register) Instruction May Result in System Hang

**Problem:** An LTR instruction may result in a system hang if all the following conditions are met:

1. Invalid data selector of the TR (Task Register) resulting with either #GP (General Protection Fault) or #NP (Segment Not Present Fault).
2. GDT (Global Descriptor Table) is not 8-bytes aligned.
3. Data BP (breakpoint) is set on cache line containing the descriptor data..

**Implication:** This erratum may result in system hang if all conditions have been met. This erratum has not been observed in commercial operating systems or software. For performance reasons, GDT is typically aligned to 8-bytes.

**Workaround:** Software should align GDT to 8-bytes

**Status:** For the steppings affected, see the Summary Tables of Changes.

### M82. Loading from Memory Type USWC (Uncacheable Speculative Write Combine) May Get Its Data Internally Forwarded from a Previous Pending Store

**Problem:** A load from memory type USWC may get its data internally forwarded from a pending store. As a result, the expected load may never be issued to the external bus.

**Implication:** When this erratum occurs, a USWC Load request may be satisfied without being observed on the external bus. There are no known usage models where this behavior results in any negative side-effects.

**Workaround:** Do not use memory type USWC for memory that has read side-effects.

**Status:** For the steppings affected, see the Summary Tables of Changes.

### M83. FXSAVE after FNINIT Without an Intervening FP (Floating Point) Instruction May Save Uninitialized Values for FDP (x87 FPU Instruction Operand (Data) Pointer Offset) and FDS (x87 FPU Instruction Operand (Data) Pointer Selector)

**Problem:** An FXSAVE after FNINIT without an intervening FP instruction may save uninitialized values for FDP and FDS.

**Implication:** When this erratum occurs, the values for FDP/FDS in the FXSAVE structure may appear to be random values. These values will be initialized by the first FP instruction executed after the FXRSTOR that restore the saved floating point state. Any FP instruction with memory operand will initialize FDP/FDS. Intel has not observed this erratum with any commercially available software.

**Workaround:** After an FINIT, do not expect the FXSAVE memory image to be correct, until at least one FP instruction with a memory operand has been executed.

**Status:** For the steppings affected, see the Summary Tables of Changes.

### M84. FSTP (Floating Point Store) Instruction Under Certain Conditions May Result In Erroneously Setting a Valid Bit on an FP (Floating Point) Stack Register

**Problem:** An FSTP instruction with an PDE/PTE (Page Directory Entry/Page Table Entry) A/D bit update followed by user mode access fault due to a code fetch to a page that has supervisor only access permission may result in erroneously setting a valid bit of an FP stack register. The FP top of stack pointer is unchanged.

**Implication:** This erratum may cause an unexpected stack overflow.

**Workaround:** User mode code should not count on being able to recover from illegal accesses to memory regions protected with supervisor only access when using FP instructions.

**Status:** For the steppings affected, see the Summary Tables of Changes.

### M85. Invalid Entries in Page-Directory-Pointer-Table Register (PDPTR) May Cause General Protection (#GP) Exception if the Reserved Bits are Set to One

**Problem:** Invalid entries in the Page-Directory-Pointer-Table Register (PDPTR) that have the reserved bits set to one may cause a General Protection (#GP) exception.

**Implication:** Intel has not observed this erratum with any commercially available software.

**Workaround:** Do not set the reserved bits to one when PDPTR entries are invalid.

§

# Specification Changes

There are no specification changes.

§

# Specification Clarifications

The Specification Clarifications listed in this section apply to:

- Mobile Intel® Celeron® Processor in BGA2 and Micro-PGA2 Packages at 900 MHz, 850 MHz, 800 MHz, 750 MHz, 700 MHz, 650 MHz, 600 MHz, 550 MHz, 500 MHz, 450 MHz, Low voltage 600 MHz, Low voltage 500 MHz, Low voltage 400A MHz, Ultra Low Voltage 600MHz and Ultra Low Voltage 500 MHz datasheet (Order Number 283654-003)
- Mobile Intel® Celeron® Processor (0.18µ) in Micro-FCBGA and Micro-FCPGA packages at 933, 866, 800A, and 733 MHz (Order Number 298514-001)
- Mobile Intel® Celeron® Processor (0.13µ) in Micro-FCBGA in Low Voltage Package at 650 MHz (Order Number 298517-001)
- Intel® Celeron® Processor Mobile Module: Mobile Module Connector 2 (MMC-2) at 700 MHz, 650 MHz, 600 MHz, 550 MHz, 500 MHz and 450 MHz datasheet
- Intel Architecture Software Developer's Manual, Volumes 1, 2, and 3
- P6 Family of Processors Hardware Developer's Manual
- Intel® Celeron® Processor – Low Power/Ultra Low Power Datasheet (Order Number 273509-01)

All Specification Clarifications will be incorporated into a future version of the appropriate documentation.

## M1. Temperature Specification Clarification for Measuring Currents

The Mobile Intel® Celeron® Processor (0.18µ) in BGA2 and Micro-PGA2 Packages Datasheet (Order Number 283654-002) has the following Specification Clarifications on note 3 of Table 32, Section 6. The temperature at which the currents are measured was not clearly specified in note 3 of Table 32.

**Table 32. Mobile Celeron Processor Power Specifications**

| Symbol | Parameter | TDP Typ[1,3] | TDP Max[2,3] | $P_{SGNT}$[3,4] | $P_{QS}$[3,5] | $P_{DSLP}$[3,6] | Unit |
|---|---|---|---|---|---|---|---|
| Power | at 500 MHz & 1.10V | 5.0 | 8.1 | 0.8 | 0.6 | 0.2 | W |
| | at 600 MHz & 1.10V | 6.4 | 9.7 | 0.8 | 0.6 | 0.2 | W |
| | at 600 MHz & 1.15V | 7.0 | 9.7 | 0.7 | 0.5 | 0.3 | W |
| | at 400A MHz & 1.35V | 6.5 | 10.1 | 1.1 | 0.8 | 0.3 | W |
| | at 500 MHz & 1.35V | 7.9 | 12.2 | 1.1 | 0.8 | 0.3 | W |
| | at 600 MHz & 1.35V | 8.7 | 14.4 | 1.1 | 0.8 | 0.3 | W |
| | at 450 MHz & 1.60V | 10.2 | 15.5 | 1.7 | 1.3 | 0.5 | W |
| | at 500 MHz & 1.60V | 11.2 | 16.8 | 1.7 | 1.3 | 0.5 | W |
| | at 550 MHz & 1.60V | 11.9 | 18.4 | 1.7 | 1.3 | 0.5 | W |
| | at 600 MHz & 1.60V | 13.0 | 20.0 | 1.7 | 1.3 | 0.5 | W |
| | at 650 MHz & 1.60V | 14.0 | 21.5 | 1.7 | 1.3 | 0.5 | W |
| | at 700 MHz & 1.60V | 15.0 | 23.0 | 2.7 | 1.9 | 0.75 | W |
| | at 750 MHz & 1.60V | 15.8 | 24.6 | 2.7 | 1.9 | 0.75 | W |
| | at 800 MHz & 1.60V | 17.6 | 25.9 | 2.51 | 1.76 | 0.87 | W |
| | at 850 MHz & 1.60V | 18.8 | 27.5 | 2.51 | 1.76 | 0.87 | W |
| $T_J$ | Junction Temperature is measured with the on-die thermal diode | 100 | 100 | 50 | 50 | 35 | °C |

**NOTE:**

1. TDPTYP is a recommendation based on the power dissipation of the processor while executing publicly available software under normal operating conditions at nominal voltages. Not 100% tested.
2. TDPMAX is a specification of the total power dissipation of the processor while executing a worst-case instruction mix under normal operating conditions at nominal voltages. It includes the power dissipated by all of the components within the processor. Not 100% tested. Specified by design/characterization.
3. Not 100% tested. These power specifications are determined by characterization of the processor currents at higher temperatures and extrapolating the values for the temperature indicated.
4. PSGNT is Stop Grant and Auto Halt power.
5. PQS is Quick Start and Sleep power.
6. PDSLP is Deep Sleep power.

## M2. SPECIFICATION CLARIFICATION WITH RESPECT TO TIME STAMP COUNTER

In the "Debugging and Performance Monitoring" section of the *IA-32 Intel® Architecture Software Developers Manual Software Developer's Manual Volume 3: System Programming Guide*, the Time Stamp Counter definition has been updated to include support for the future processors. This change will be incorporated in the next revision of the *IA-32 Intel® Architecture Software Developers Manual Software Developer's Manual Volume 3: System Programming Guide*.

IA-32 Intel® Architecture Software Developers Manual Software Developer's Manual Volume 3: System Programming Guide Debugging and Performance Monitoring Section (Section 15.8, Section 15.10.9 and Section 15.10.9.3)

### 15.8 TIME-STAMP COUNTER

The IA-32 architecture (beginning with the Pentium processor) defines a time-stamp counter mechanism that can be used to monitor and identify the relative time occurrence of processor events. The counter's architecture includes the following components:

- **TSC flag —** A feature bit that indicates the availability of the time-stamp counter. The counter is available in an IA-32 processor implementation if the function CPUID.1:EDX.TSC[bit 4] = 1.
- **IA32_TIME_STAMP_COUNTER MSR** (called TSC MSR in P6 family and Pentium processors) **—** The MSR used as the counter.
- **RDTSC instruction —** An instruction used to read the time-stamp counter.
- **TSD flag —** A control register flag is used to enable or disable the time-stamp counter (enabled if CR4.TSD[bit 2] = 1).

The time-stamp counter (as implemented in the P6 family, Pentium, Pentium M, Pentium 4, and Intel Xeon processors) is a 64-bit counter that is set to 0 following a RESET of the processor. Following a RESET, the counter will increment even when the processor is halted by the HLT instruction or the external STPCLK# pin. Note that the assertion of the external DPSLP# pin may cause the time-stamp counter to stop.

Members of the processor families increment the time-stamp counter differently:

- For Pentium M processors (family [06H], models [09H, 0DH]); for Pentium 4 processors, Intel Xeon processors (family [0FH], models [00H, 01H, or 02H]); and for P6 family processors: the time-stamp counter increments with every internal processor clock cycle. The internal processor clock cycle is determined by the current core-clock to bus-clock ratio. Intel SpeedStep® technology transitions may also impact the processor clock.
- For Pentium 4 processors, Intel Xeon processors (family [0FH], models [03H and higher]): the time-stamp counter increments at a constant rate. That rate may be set by the maximum core-clock to bus-clock ratio of the processor or may be set by the frequency at which the processor is booted. The specific processor configuration determines the behavior. Constant TSC behavior ensures that the duration of each clock tick is uniform and supports the use of the TSC as a wall clock timer even if the processor core changes frequency. This is the architectural behavior moving forward.

NOTE

To determine average processor clock frequency, Intel recommends the use of Performance Monitoring logic to count processor core clocks over the period of time

for which the average is required. See Section 15.10.9 and Appendix A in this manual for more information.

The RDTSC instruction reads the time-stamp counter and is guaranteed to return a monotonically increasing unique value whenever executed, except for a 64-bit counter wraparound. Intel guarantees that the time-stamp counter will not wraparound within 10 years after being reset. The period for counter wrap is longer for Pentium 4, Intel Xeon, P6 family, and Pentium processors.

Normally, the RDTSC instruction can be executed by programs and procedures running at any privilege level and in virtual-8086 mode. The TSD flag allows use of this instruction to be restricted to programs and procedures running at privilege level 0. A secure operating system would set the TSD flag during system initialization to disable user access to the time-stamp counter. An operating system that disables user access to the time-stamp counter should emulate the instruction through a user-accessible programming interface.

The RDTSC instruction is not serializing or ordered with other instructions. It does not necessarily wait until all previous instructions have been executed before reading the counter. Similarly, subsequent instructions may begin execution before the RDTSC instruction operation is performed.

The RDMSR and WRMSR instructions read and write the time-stamp counter, treating the time-stamp counter as an ordinary MSR (address 10H). In the Pentium 4, Intel Xeon, and P6 family processors, all 64-bits of the time-stamp counter are read using RDMSR (just as with RDTSC). When WRMSR is used to write the time-stamp counter on processors before family [0FH], models [03H, 04H]: only the low order 32-bits of the time-stamp counter can be written (the high-order 32 bits are cleared to 0). For family [0FH], models [03H, 04H]: all 64 bits are writeable.

### 15.10.9 COUNTING CLOCKS

The count of cycles, also known as clockticks, forms a the basis for measuring how long a program takes to execute. Clockticks are also used as part of efficiency ratios like cycles per instruction (CPI). Processor clocks may stop ticking under circumstances like the following:

- The processor is halted when there is nothing for the CPU to do. For example, the processor may halt to save power while the computer is servicing an I/O request. When Hyper-Threading Technology† is enabled, both logical processors must be halted for performance-monitoring counters to be powered down.
- The processor is asleep as a result of being halted or because of a power-management scheme. There are different levels of sleep. In the some deep sleep levels, the time-stamp counter stops counting.

There are three ways to count processor clock cycles to monitor performance. These are:

- **Non-halted clockticks —** Measures clock cycles in which the specified logical processor is not halted and is not in any power-saving state. When Hyper-Threading Technology is enabled, this these ticks can be measured on a per-logical-processor basis.
- **Non-sleep clockticks —** Measures clock cycles in which the specified physical processor is not in a sleep mode or in a power-saving state. These ticks can**n**ot be measured on a logical-processor basis.
- **Time-stamp counter —** Measures clock cycles in which the physical processor is not in deep sleep. These ticks cannot be measured on a logical-processor basis.

- **Time-stamp counter** — Some processor models permit clock cycles to be measured when the physical processor is not in deep sleep (by using the time-stamp counter and the RDTSC instruction). Note that such sticks cannot be measured on a per-logical-processor basis. See Section 10.8 for detail on processor capabilities.

The first two methods use performance counters and can be set up to cause an interrupt upon overflow (for sampling). They may also be useful where it is easier for a tool to read a performance counter than to use a time stamp counter (the timestamp counter is accessed using the RDTSC instruction).

For applications with a significant amount of I/O, there are two ratios of interest:

- **Non-halted CPI —** Non-halted clockticks/instructions retired measures the CPI for phases where the CPU was being used. This ratio can be measured on a logical-processor basis when Hyper-Threading Technology is enabled.
- **Nominal CPI —** Time-stamp counter ticks/instructions retired measures the CPI over the duration of a program, including those periods when the machine halts while waiting for I/O.

#### 15.10.9.3 Incrementing the Time-Stamp Counter

The time-stamp counter increments when the clock signal on the system bus is active and when the sleep pin is not asserted. The counter value can be read with the RDTSC instruction.
The time-stamp counter and the non-sleep clockticks count may not agree in all cases and for all processors. See Section 10.8 for more information on counter operation.

§

# Documentation Changes

There are no Documentation Changes for this month.

§